(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 December 2000 (21.12.2000)

PCT

(10) International Publication Number
WO 00/77750 A1

(51) International Patent Classification[7]: G07F 7/10, G06F 11/00

(21) International Application Number: PCT/CA00/00703

(22) International Filing Date: 14 June 2000 (14.06.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/138,679 14 June 1999 (14.06.1999) US

(71) Applicant *(for all designated States except US)*: AUDESI TECHNOLOGY INC. [CA/CA]; Suite 108, 6815 8th Street, N.E., Calgary, Alberta T2E 7H7 (CA).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: MARYKA, Stephen [CA/CA]; 292 Hawkwood Drive, N.W., Calgary, Alberta T3G 3N9 (CA). MICHAUD, Bertrand [CA/CA]; 27 Beddington Green, N.E., Calgary, Alberta T3K 1M7 (CA).

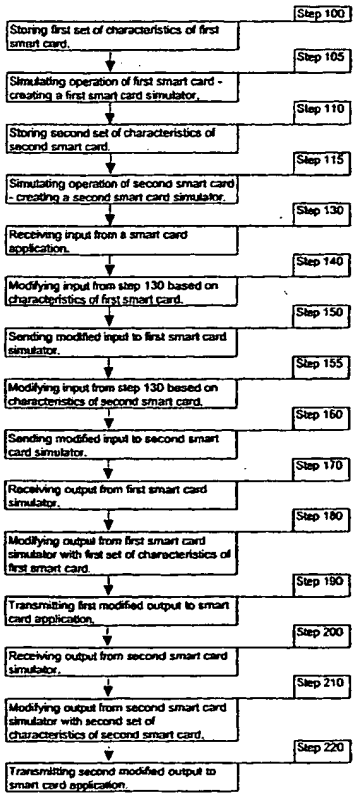
(74) Agents: LIMPERT, P., Brad et al.; Gowling Lafleur Henderson LLP, Suite 4900, Commerce Court West, Toronto, Ontario M5L 1J3 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,

[Continued on next page]

(54) Title: METHOD FOR A MULTIPLE SMART CARD SIMULATOR



(57) Abstract: The invention is a method for a multiple smart card simulator. The invention allows cards with different characteristics such as memory, command sequence, error handling, and data flow to be simultaneously simulated. It also permits simultaneous simulation of different smart card user applications. The method involves determining the characteristics of smart cards, and then modifying input and output to or from these cards based on the smart card characteristics. In this way, an end-to-end simulation environment is provided.

WO 00/77750 A1

IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *With international search report.*

— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Title: **METHOD FOR A MULTIPLE SMART CARD SIMULATOR**

This application claims priority from U.S. provisional application 60/138,679 filed June 14, 1999.

Field of Invention

The present application relates to a method for a multiple smart card simulator, and in particular, an application employing multiple simulated smart-cards where the applications running on, or enabled on, some of the smart cards differ from those applications running on, or enabled on, other of the simulated smart cards, and where the smart cards may have different characteristics.

Background to Invention

Smart cards are hardware devices that typically have the shape and size of a credit card. Often they have an embedded computer processor, memory and an operating system. Often smart cards have stored in their memory information related to the user of the smart card. Such information can include:

(i) personal information, such as name, address, other contact information, date of birth, etc.;

(ii) financial information, such as bank account numbers and type of account, account balances and transaction histories, retirement savings plan information, investment information and investment portfolio and account information;

(iii) medical information, such as medical history and known allergies and prescription history;

(iv) information related to retail shopping, such as clothing sizes, shopping histories and product, preferences; and

(v) other information relating to taste, style and economic, social or cultural activity.

Smart cards are inserted into card terminals which may include automated teller machines, internet kiosks, pay telephones, point of sale terminals, cellular phones, or any device containing a card reader.

Using the information stored on the smart card, the card terminals provide a variety of services and computer-based applications to the user of the smart card. These services can include:

(i) paying bills, transferring money between accounts, making investments, analyzing a portfolio;

(ii) ordering prescription drugs, providing information to health care professionals or receiving information from them;

(iii) buying or selling consumer goods, receiving product information, participating in consumer surveys, receiving promotional offers, customizing products; and

(iv) playing games or receiving other content such as music, video, text or images.

Different users of smart cards may want to use different applications for a variety of reasons: smart cards may have different capabilities, users may have different needs and preferences, users may be authorized to use different applications.

Developers of smart card applications often prefer to develop these applications in a simulated environment. In the simulated environment, many hardware elements of the system such as the smart card, cardreader and card terminal are simulated. This allows the developer to test and develop function, eliminate error, and improve performance of the system without the cumbersome and costly need to implement all of the hardware elements.

Current simulation environments, such as the Java Card™ simulation environment suffer the drawback that they are unable to simulate an application that deals with multiple smart cards where those smart cards have different applications or where the smart cards have different characteristics.

Summary of Invention

The purpose of the present invention is to provide a method and apparatus for simulating a smart-card-based computer application.

An advantage of the invention is that it supports multi-application card environments. Another advantage of the present invention is that it allows simultaneous simulation of smart cards with different characteristics. A further advantage is that it allows simulation of the smart card, card reader, terminal, and user application in a single development workstation.

In one embodiment of the present invention there is provided a method for a multiple smart card simulator.

Description of Figures

Figure 1 shows a flow chart describing an embodiment for the present invention;

Figure 2 shows a flow chart describing an embodiment for the present invention;

Figure 3 shows a flow chart describing an embodiment for the present invention.

Detailed Description

Smart cards often have different characteristics. These characteristics include available memory, the manner in which they handle error conditions, the command sequence to be input or the data returned from a command sequence. Smart cards often also have idiosyncratic quirks or bugs. These characteristics and quirks must be simulated in order to have an effective simulation.

The present invention provides a method for providing a multiple smart card simulator. In one embodiment, as set out in figure 1, the following steps occur:

(i) storing a first set of characteristics of a first smart card (step 100). These are stored in memory in the developer's workstation;

(ii) creating a first smart card simulator simulating operation of said first smart card (step 105). The simulator operates by processing data in the same way that a real smart card would;

(iii) storing a second set of characteristics of a second smart card (Step 110);

(iv) simulating operation of said second smart card thereby creating a second smart card simulator (step 115). The simulator operates by processing data in the same way that a real smart card would;

(v) receiving a first input from a first smart card application (step 130). As used herein, smart card applications refer to user applications running on the developer's work station or on a simulated terminal and not to thin or trivial applications running only on the smart card;

(vi) modifying said first input based on said first set of characteristics (step 140);

(vii) sending a first modified input to said first smart card simulator (step 150);

(viii) modifying said first input based on such second set of characteristics to form a second modified input; and sending the second modified input to said second smart card simulator (steps 155 and 160). The second simulator needs a different modified input because it has different characteristics;

(ix) receiving a first output from said first smart card simulator generated in response to said first modified input (step 170). The output is generated by the simulator by methods known to those skilled in the art;

(x) modifying said first output based on said first set of characteristics to form a first modified output (step 180);

(xi) transmitting said first modified output to said smart card application (step 190);

(xii) receiving a second output from said second smart card simulator generated in response to said second modified input (step 200). The output is generated by means to those skilled in the art;

(xiii) modifying said second output based on said second set of characteristics to form a second modified output (step 210);

(xiv) transmitting said second modified output to said smart card application (step 220);

In another embodiment, a method is provided for simultaneously running different applications on different smart cards. This method is shown on Figure 2 and comprises the following steps:

(i) receiving a second input from a second smart card application (step 300);

(ii) modifying said second input based on said first set of characteristics (step 310);

(iii) sending a third modified input to said first smart card simulator (step 320);

(iv) receiving a third output from said first smart card simulator in response to said third modified input (step 330);

(v) modifying first said third output based on said first set of characteristics (step 340);

(vi) transmitting said third modified output to said second smart card application (step 350);

One important aspect of the invention is the ability to simulate card insertion. Depending on the characteristics of the inserted card, different applications may be downloaded into the card. This method is shown in Figure 3 and comprises the steps:

(i) generating a simulated smart card insertion (step 400);

(ii) determining characteristics of the inserted card (step 405);

(iii) notifying registered listeners (step 410);

(iv) determining available applications (step 420);

(v) downloading to a simulated smart card and to a simulated terminal at least one smart card application (step 430) and

(vi) repeating (steps 400 - 430)

In the above methods it may be advantageous to simulate Java Cards, for which many development tools currently exist and for which the same program code can run in the actual smart card and in the simulator, namely the Java programming language.

Further details are set out in appendix A "Alchemy: Science and Magic for Intelligent Devices" especially sections 5 and 5.1 and in co-pending U.S. applications 09/332,069 titled "Method and Apparatus for Incremental Download from Server to Client", 09/332,191 titled "Method and System of Deploying an Application between Computers", 09/332,192 titled "Method and System for Remotely Observing and Controlling Objects and 09/332,193 titled "Method and System for Managing and Using Persistent Storage all of which are incorporated by reference.

We claim:

1. A method of providing a multiple smart card simulator comprising:

(i) storing a first set of characteristics of a first smart card;

(ii) creating a first smart card simulator simulating operation of said first smart card;

(iii) storing a second set of characteristics of a second smart card;

(iv) simulating operation of said second smart card thereby creating a second smart card simulator;

(v) receiving a first input from a first smart card application;

(vi) modifying said first input based on said first set of characteristics;

(vii) sending a first modified input to said first smart card simulator;

(viii) modifying said first input based on such second set of characteristics to form a second modified input; and sending the second modified input to said second smart card simulator;

(ix) receiving a first output from said first smart card simulator generated in response to said first modified input;

(x) modifying said first output based on said first set of characteristics to form a first modified output;

(xi) transmitting said first modified output to said smart card application;

(xii) receiving a second output from said second smart card simulator in response to said second modified input;

(xiii) modifying said second output based on said second set of characteristics to form a second modified output; and

(xiv) transmitting said second modified output to said smart card application.

2. The method of claim 1 where said characteristics include one of the smart cards memory size, command sequence, error handling routine or quirks.

3. The method of claim 1 further comprising the steps:

(i) receiving a second input from a second smart card application;

(ii) modifying said second input based on said first set of characteristics;

(iii) sending a third modified input to said first smart card simulator;

(iv) receiving a third output from said first smart card simulator in response to said third modified input;

(v) modifying first said third output based on said first set of characteristics;

(vi) transmitting said third modified output to said second smart card application.

4. A method of simulating a multiple smart card environment comprising:

(i) detecting a simulated smart card insertion (step 400);

(ii) determining characteristics of the inserted card (step 405);

(iii) notifying registered listeners (step 410);

(iv) determining available applications (step 420);

(v) downloading to a simulated card and to a simulated terminal at least one smart card application (step 430) and

(vi) repeating (steps 400 - 430).

Figure 1

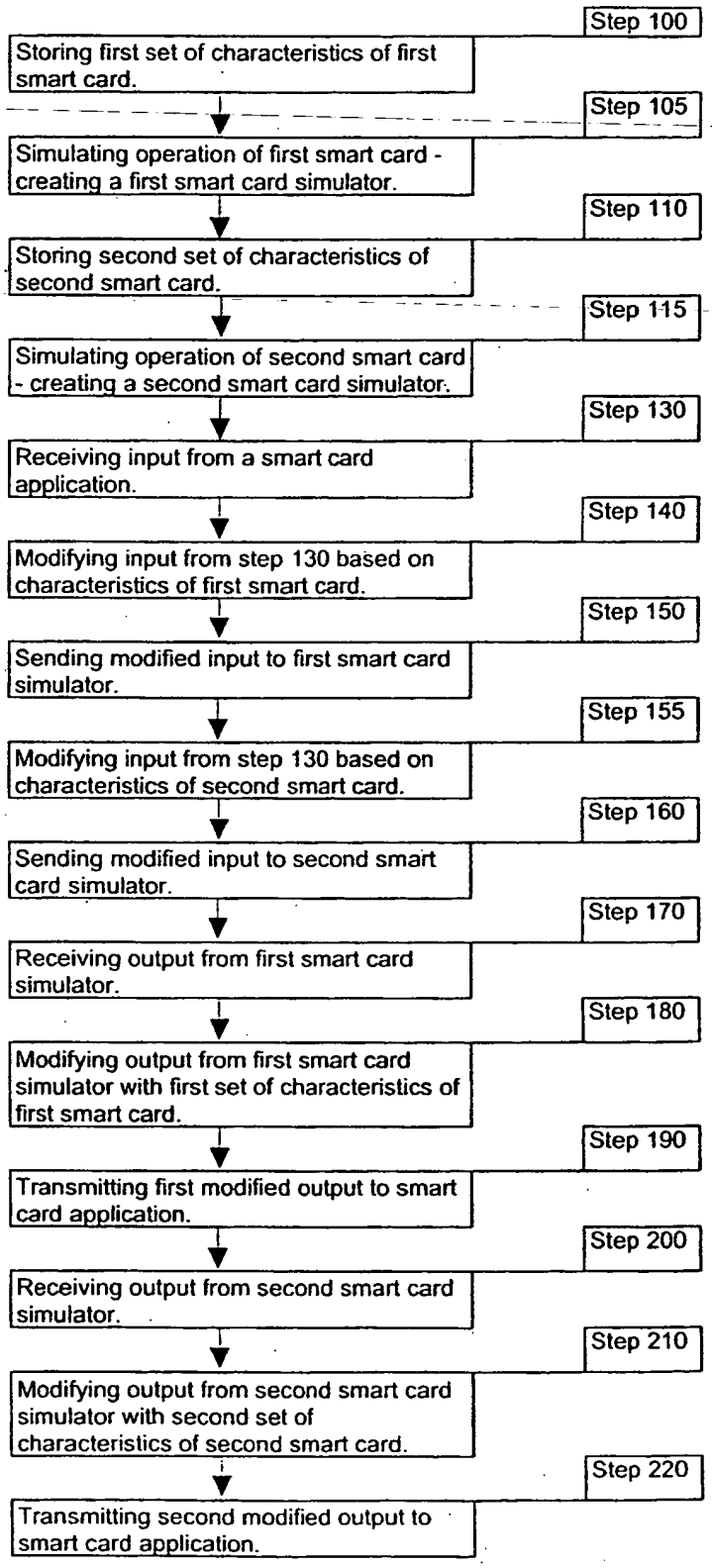

Step 100
Storing first set of characteristics of first smart card.
Step 105
Simulating operation of first smart card - creating a first smart card simulator.
Step 110
Storing second set of characteristics of second smart card.
Step 115
Simulating operation of second smart card - creating a second smart card simulator.
Step 130
Receiving input from a smart card application.
Step 140
Modifying input from step 130 based on characteristics of first smart card.
Step 150
Sending modified input to first smart card simulator.
Step 155
Modifying input from step 130 based on characteristics of second smart card.
Step 160
Sending modified input to second smart card simulator.
Step 170
Receiving output from first smart card simulator.
Step 180
Modifying output from first smart card simulator with first set of characteristics of first smart card.
Step 190
Transmitting first modified output to smart card application.
Step 200
Receiving output from second smart card simulator.
Step 210
Modifying output from second smart card simulator with second set of characteristics of second smart card.
Step 220
Transmitting second modified output to smart card application.

Figure 2

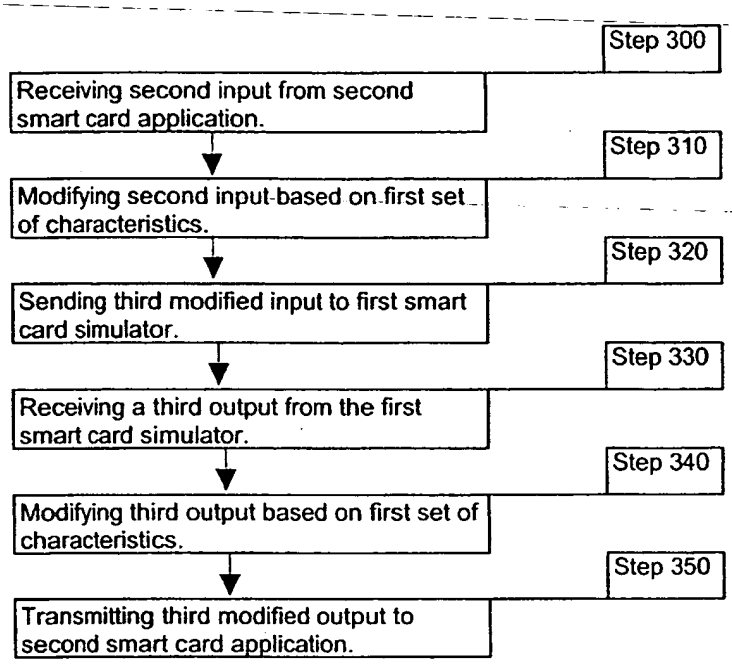
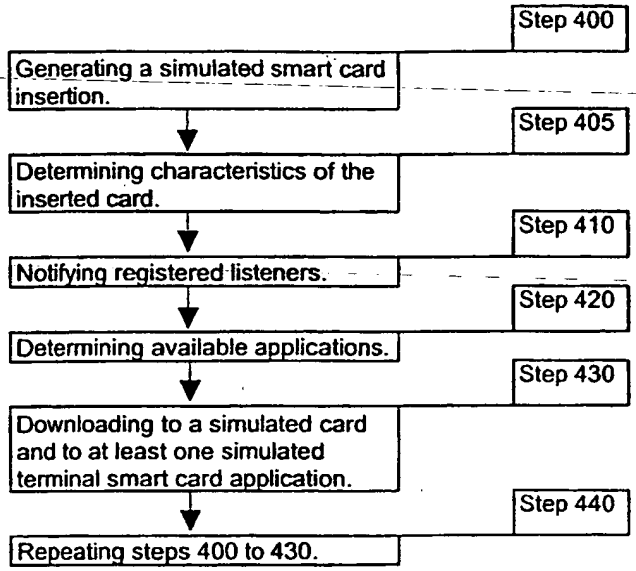

Step 300
Receiving second input from second smart card application.
Step 310
Modifying second input based on first set of characteristics.
Step 320
Sending third modified input to first smart card simulator.
Step 330
Receiving a third output from the first smart card simulator.
Step 340
Modifying third output based on first set of characteristics.
Step 350
Transmitting third modified output to second smart card application.

Figure 3


Step 400
Generating a simulated smart card insertion.
Step 405
Determining characteristics of the inserted card.
Step 410
Notifying registered listeners.
Step 420
Determining available applications.
Step 430
Downloading to a simulated card and to at least one simulated terminal smart card application.
Step 440
Repeating steps 400 to 430.

INTERNATIONAL SEARCH REPORT

Intern: al Application No
PCT/CA 00/00703

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G07F7/10 G06F11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G07F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 98 25239 A (STRATEGIC ANALYSIS INC) 11 June 1998 (1998-06-11) page 2, line 35 -page 3, line 17 page 4, line 12 - line 26 page 5, line 7 - line 29 page 8, line 27 -page 9, line 8 page 10, line 12 -page 11, line 9	1-4
X	FR 2 667 419 A (GEMPLUS CARD INT) 3 April 1992 (1992-04-03) page 2, line 15 -page 4, line 1	1-4
A	PATENT ABSTRACTS OF JAPAN vol. 014, no. 141 (P-1023), 16 March 1990 (1990-03-16) & JP 02 005191 A (FUJITSU LTD), 10 January 1990 (1990-01-10) abstract	

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
6 November 2000	15/11/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Wolles, B

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

...ormation on patent family members

Intern: al Application No
PCT/CA 00/00703

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9825239 A	11-06-1998	AU 5595398 A	29-06-1998
		EP 0943136 A	22-09-1999
FR 2667419 A	03-04-1992	NONE	
JP 02005191 A	10-01-1990	JP 2852381 B	03-02-1999

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 December 2000 (21.12.2000)

PCT

(10) International Publication Number
WO 00/077750 A1

(51) International Patent Classification[7]: G07F 7/10, G06F 11/00

(21) International Application Number: PCT/CA00/00703

(22) International Filing Date: 14 June 2000 (14.06.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/138,679 14 June 1999 (14.06.1999) US

(71) Applicant *(for all designated States except US)*: **WIND RIVER INTERNATIONAL LIMITED** [CA/CA]; Suite 108, 6815 8th Street, N.E., Calgary, Alberta T2E 7H7 (CA).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **MARYKA, Stephen** [CA/CA]; 292 Hawkwood Drive, N.W., Calgary, Alberta T3G 3N9 (CA). **MICHAUD, Bertrand** [CA/CA]; 27 Beddington Green, N.E., Calgary, Alberta T3K 1M7 (CA).

(74) Agents: **LIMPERT, P., Brad** et al.; Gowling Lafleur Henderson LLP, Suite 4900, Commerce Court West, Toronto, Ontario M5L 1J3 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(48) Date of publication of this corrected version:
29 August 2002

(15) Information about Correction:
see PCT Gazette No. 35/2002 of 29 August 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD FOR A MULTIPLE SMART CARD SIMULATOR

(57) Abstract: The invention is a method for a multiple smart card simulator. The invention allows cards with different characteristics such as memory, command sequence, error handling, and data flow to be simultaneously simulated. It also permits simultaneous simulation of different smart card user applications. The method involves determining the characteristics of smart cards, and then modifying input and output to or from these cards based on the smart card characteristics. In this way, an end-to-end simulation environment is provided.

WO 00/077750 A1

METHOD FOR A MULTIPLE SMART CARD SIMULATOR

This application claims priority from U.S. provisional application 60/138,679 filed June 14, 1999.

Field of Invention

The present application relates to a method for a multiple smart card simulator, and in particular, an application employing multiple simulated smart-cards where the applications running on, or enabled on, some of the smart cards differ from those applications running on, or enabled on, other of the simulated smart cards, and where the smart cards may have different characteristics.

Background to Invention

Smart cards are hardware devices that typically have the shape and size of a credit card. Often they have an embedded computer processor, memory and an operating system. Often smart cards have stored in their memory information related to the user of the smart card. Such information can include:

(i) personal information, such as name, address, other contact information, date of birth, etc.;

(ii) financial information, such as bank account numbers and type of account, account balances and transaction histories, retirement

savings plan information, investment information and investment portfolio and account information;

(iii) medical information, such as medical history and known allergies and prescription history;

(iv) information related to retail shopping, such as clothing sizes, shopping histories and product, preferences; and

(v) other information relating to taste, style and economic, social or cultural activity.

Smart cards are inserted into card terminals which may include automated teller machines, internet kiosks, pay telephones, point of sale terminals, cellular phones, or any device containing a card reader.

Using the information stored on the smart card, the card terminals provide a variety of services and computer-based applications to the user of the smart card. These services can include:

(i) paying bills, transferring money between accounts, making investments, analyzing a portfolio;

(ii) ordering prescription drugs, providing information to health care professionals or receiving information from them;

(iii) buying or selling consumer goods, receiving product information, participating in consumer surveys, receiving promotional offers, customizing products; and

(iv) playing games or receiving other content such as music, video, text or images.

Different users of smart cards may want to use different applications for a variety of reasons: smart cards may have different capabilities, users may have different needs and preferences, users may be authorized to use different applications.

Developers of smart card applications often prefer to develop these applications in a simulated environment. In the simulated environment, many hardware elements of the system such as the smart card, cardreader and card terminal are simulated. This allows the developer to test and develop function, eliminate error, and improve performance of the system without the cumbersome and costly need to implement all of the hardware elements.

Current simulation environments, such as the Java Card™ simulation environment suffer the drawback that they are unable to simulate an application

that deals with multiple smart cards where those smart cards have different applications or where the smart cards have different characteristics.

Summary of Invention

The purpose of the present invention is to provide a method and apparatus for simulating a smart-card-based computer application.

An advantage of the invention is that it supports multi-application card environments. Another advantage of the present invention is that it allows simultaneous simulation of smart cards with different characteristics. A further advantage is that it allows simulation of the smart card, card reader, terminal, and user application in a single development workstation.

In one embodiment of the present invention there is provided a method for a multiple smart card simulator.

Description of Figures

Figure 1 shows a flow chart describing an embodiment for the present invention;

Figure 2 shows a flow chart describing an embodiment for the present invention;

Figure 3 shows a flow chart describing an embodiment for the present invention.

Detailed Description

Smart cards often have different characteristics. These characteristics include available memory, the manner in which they handle error conditions, the command sequence to be input or the data returned from a command sequence. Smart cards often also have idiosyncratic quirks or bugs. These characteristics and quirks must be simulated in order to have an effective simulation.

The present invention provides a method for providing a multiple smart card simulator. In one embodiment, as set out in figure 1, the following steps occur:

(i) storing a first set of characteristics of a first smart card (step 100). These are stored in memory in the developer's workstation;

(ii) creating a first smart card simulator simulating operation of said first smart card (step 105). The simulator operates by processing data in the same way that a real smart card would;

(iii) storing a second set of characteristics of a second smart card (Step 110);

(iv) simulating operation of said second smart card thereby creating a second smart card simulator (step 115). The simulator operates by processing data in the same way that a real smart card would;

(v) receiving a first input from a first smart card application (step 130). As used herein, smart card applications refer to user applications running on the developer's work station or on a simulated terminal and not to thin or trivial applications running only on the smart card;

(vi) modifying said first input based on said first set of characteristics (step 140);

(vii) sending a first modified input to said first smart card simulator (step 150);

(viii) modifying said first input based on such second set of characteristics to form a second modified input; and sending the second modified input to said second smart card simulator (steps 155 and 160). The second simulator needs a different modified input because it has different characteristics;

(ix) receiving a first output from said first smart card simulator generated in response to said first modified input (step 170). The

output is generated by the simulator by methods known to those skilled in the art;

(x) modifying said first output based on said first set of characteristics to form a first modified output (step 180);

(xi) transmitting said first modified output to said smart card application (step 190);

(xii) receiving a second output from said second smart card simulator generated in response to said second modified input (step 200). The output is generated by means to those skilled in the art;

(xiii) modifying said second output based on said second set of characteristics to form a second modified output (step 210);

(xiv) transmitting said second modified output to said smart card application (step 220);

In another embodiment, a method is provided for simultaneously running different applications on different smart cards. This method is shown on Figure 2 and comprises the following steps:

(i) receiving a second input from a second smart card application (step 300);

(ii) modifying said second input based on said first set of characteristics (step 310);

(iii) sending a third modified input to said first smart card simulator (step 320);

(iv) receiving a third output from said first smart card simulator in response to said third modified input (step 330);

(v) modifying first said third output based on said first set of characteristics (step 340);

(vi) transmitting said third modified output to said second smart card application (step 350);

One important aspect of the invention is the ability to simulate card insertion. Depending on the characteristics of the inserted card, different applications may be downloaded into the card. This method is shown in Figure 3 and comprises the steps:

(i) generating a simulated smart card insertion (step 400);

(ii) determining characteristics of the inserted card (step 405);

(iii) notifying registered listeners (step 410);

(iv) determining available applications (step 420);

(v) downloading to a simulated smart card and to a simulated terminal at least one smart card application (step 430) and

(vi) repeating (steps 400 - 430)

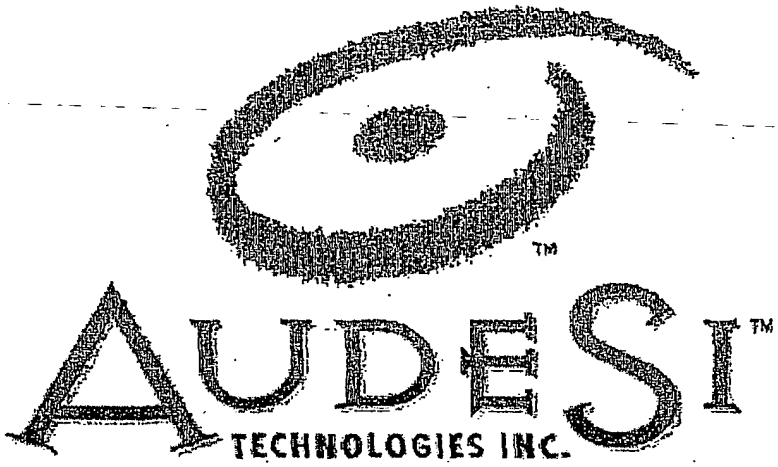
In the above methods it may be advantageous to simulate Java Cards, for which many development tools currently exist and for which the same program code can run in the actual smart card and in the simulator, namely the Java programming language.

Further details are set out in appendix A "Alchemy: Science and Magic for Intelligent Devices" especially sections 5 and 5.1 and in co-pending U.S. applications 09/332,069 titled "Method and Apparatus for Incremental Download from Server to Client", 09/332,191 titled "Method and System of Deploying an Application between Computers", 09/332,192 titled "Method and System for Remotely Observing and Controlling Objects and 09/332,193 titled "Method and

System for Managing and Using Persistent Storage all of which are incorporated by reference.

5

APPENDIX A



Alchemy™:

Science and Magic

For Intelligent Devices

Table of Contents

Table of Figures

Touchstone™ and Alchemy™ are registered trademarks of AudeSi Technologies Inc.

Windows, Windows NT, and Windows 95 are registered trademarks of Microsoft Corporation in the United States and/or other countries.

PowerPC is a registered trademark of International Business Machines Corporation.

GCR 410 is a registered trademark of Gemplus.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc, in the United States and/or other countries.

JWorks™ is a registered trademark of WindRiver Systems.

All other trademarks used in this document are the property of their respective owners.

Acronyms

Acronym	Description
ADPCM	Adaptive Differential Pulse Code Modulation
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATA	Advanced Technology Attachment
AWT	Abstract Windowing Toolkit
CLID	Calling Line ID
CPM	Communication Processor Module
CSO	Chip Select O
DNS	Domain Name Server
DSP	Digital Signal Processing
DRAM	Dynamic Random Access Memory
DSVD	Digital Simultaneous Voice Data
DTAD	Digital Telephone Answering Device
DTMF	Dual Tone Multi Frequency
EEPROM	Electronically Erasable Programmable Read Only Memory
EFS	Flash File System
FSK	Frequency Shift Keying
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hyper Text Transmission Protocol
HTTPS	Hyper Text Transmission Protocol - Secure Socket Layer (HTTP-SSL)
IDE	Integrated Development Environment
I/O	Input/Output
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
ISP	Internet Service Provider
JAR	Java Archive Storage
JNI	Java Native Interface
JVM	Java Virtual Machine
Kbps	Kilo Bites Per Second
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LIFO	Last In, First Out
NAND Flash	Negative AND Flash
NOR Flash	Negative OR Flash

Acronym	Description
OCF	Open Card Framework
O&C	Observation and Control
PDA	Personal Digital Assistant
PIM	Personal Information Manager
POTS	Plain Old Telephone Service
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network
RAD	Rapid Application Development
ROM	Read Only Memory
RTOS	Real Time Operating System
SDRAM	Synchronous Dynamic Random Access Memory
SCWID	Spontaneous Caller Waiting ID
TCP/IP	Transmission Control Protocol/Internet Protocol
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
VM	Virtual Machine

1 Introduction

Advances in Internet, telephony, and other communications technologies have spurred demand for a wide range of new, intelligent devices. Armed with these devices, service providers will soon offer a host of new Internet services to a new class of users. Easy access to services will fuel the continuing explosion of the Internet, ultimately creating greater demand for intelligent access devices. As a device manufacturer, this is the kind of chain reaction you want to be right in the middle of... but how do you get in the game? Many devices will be deployed including screen phones, counter-top tablets, set top boxes, mobile phones, public kiosks, Personal Digital Assistants (PDA), etc. Regardless of the device, there are a couple of things to keep in mind, if you are going to be successful.

1. Your device is going to have to be "slick" so you can differentiate yourself from the competition.
2. The time to market is critical so you will want to compress your development time.
3. Development dollars are scarce so use them where they count the most, on the value add content.
4. People that know how to put Java™ into embedded appliances are very scarce. Work on getting them now.
5. The Internet is bigger than you are, so you need a way to integrate the rapidly emerging technologies without getting buried.

What will it take to get this device developed?

First, you need a hardware solution. You can probably find a reference design as a starting point, but you're going to need to build a development board with the proper features for your device. You need to get started right away because it will take 4 to 6 months. Beyond that timeframe, you will also have to spin a form factor board, which will take another 4 to 6 months.

Next, you need a proper environment and strategy for software development. You need to develop the software parallel to the hardware because you won't see the hardware for several months. Java™ is a good choice for achieving platform independence and it has "network awareness" within it. Good news comes from your Real Time Operating System (RTOS) vendor because he has an embeddable Java solution you can deploy on your hardware when it's ready.

It's not a bed of roses yet... you have a full suite of device drivers to write before integrating your software and hardware. Better get your RTOS engineers working right away as driver development might end up on the critical path.

Now, let's move to the application suite. The initial set of applications to be deployed is well defined but the dynamic nature of the service environment will inevitably give rise to new requirements. You need a solid multi-application infrastructure that allows you to react to changing requirements... but there isn't time to do it properly for this device.

So, you take shortcuts and begin to cobble together your application suite. Debugging proves to be more complicated than expected because the software is multi-threaded. It would be nice to have debugging tools tailored to your architecture, but you don't have time or resources to develop tools. You struggle with traditional tools and some rudimentary tracing facilities.

Eventually, you have a "mostly working" application suite and hardware platform. You are ready to begin integration but progress is slow. It proves to be non-trivial to configure a target device properly and get a Java™ software load onto it. It would be nice to have a seamless environment covering the entire spectrum from the Java™ application suite to RTOS and drivers... but you don't. Debugging on the target is even more challenging than on the workstation. You probably should have developed those test tools but it's too late now.

Finally, your applications are running on the target and are close to achieving the functional requirements for the device... but you aren't out of the woods yet. Your boot time is too slow and your memory requirements are too large. You need to re-architect your application suite to improve the initialization sequence. Meanwhile, you find tools that can help trim down your Java™ load, but they are not well integrated and prove to be cumbersome. You make the tools work for you, but you really need to address these environmental issues before your next device.

You are ready for field trials, but you had to pull all your diagnostic capabilities to achieve a representative software load. You are now flying blind when it comes to debugging anomalies in the field. It would be great to have support for remote management and monitoring of the device. You would double benefit from this when it comes time to performing automatic regression testing. Put it on the wish list for next time... this device is done!

It took 18 months to deliver, and cost you over 96 man-months of software effort and 48 man-months of hardware resources. You hope you haven't missed your window of opportunity. The next device will go a lot smoother now that you've been through it once. At least you hope so! Don't let this description become a brief history of your development effort. There is a better way.

Alchemy™

Derived from real world experience gained in developing two generations of Java-based intelligent devices, AudeSi Technologies Inc. delivers **Alchemy™**, a unique software and hardware solution targeted at rapid development of intelligent devices requiring Internet, telephony, and/or smart card capabilities.

Alchemy™ includes a Java-based multi-application software framework, a variety of plug-in software services, working example applications, and a comprehensive development and test environment that addresses the full spectrum of device-level software development issues. Alchemy™, when integrated with Touchstone™ and associated drivers, provides a superset of hardware capabilities required for intelligent devices. This includes a PowerPC™ based computation engine, multiple memory configurations, an advanced two-line telephony block, smart card readers, and a host of I/O capabilities including Ethernet, modem, serial, and universal serial bus (USB). Details on Touchstone™ are included in Appendix A.

The primary goal of Alchemy™ is to reduce the barriers to entry for prospective manufacturers and developers of intelligent devices. Alchemy™ achieves these goals by providing the following.

1. A Java Native Interface (JNI) provides access to Touchstone™ hardware platform capabilities from the Java™ layer, thus eliminating the need for developers to work with low-level device drivers and RTOS issues.
2. The Alchemy™ Core Framework provides a comprehensive environment for the deployment of an extensible multi-application software suite. The framework handles a variety of low-level issues including device configuration, initialization sequence, and memory/file-system management, thus allowing the developer to concentrate on application level development.
3. The framework is "Internet Aware" and supports seamless integration with network-based server environments. Two primary capabilities delivered through network awareness are as follows:
    - Remote device management where a server-based application can manage a device for the purpose of device configuration, software upgrade, or in-field diagnostics.
    - Distributed framework services where applications can be seamlessly partitioned across the device/server environment, ultimately reducing device resource requirements by offloading portions of the application onto server resources.
4. Alchemy™ is based on the JavaBeans™ programming model, and as such, facilitates development of portable and reusable software components, and integration with commercially available Java™ software development tools and environments.
5. Alchemy™ delivers comprehensive services in support of application development including the following:
    - Application services including application registration, application selection, language management, and user preference management.
    - Internet services including point-to-point protocol (PPP) and integration of 3rd party browsers and email clients.
    - Smart card services including Open Card Framework™ (OCF) and Java Card™ support.

- Telephony services including voice call control state machine, dual tone multi frequency (DTMF) tone generation, frequency shift keying (FSK) data reception, Calling Line ID/Spontaneous Caller Waiting ID (CLID/SCWID), contact database, and modem control.
- Observation and control services that facilitate remote testing of devices.
- Device Management Gateway services for secured management of device configuration and application download.

Beyond this initial set of services, Alchemy™ will be continually enhanced with new service offerings in a variety of areas including security and cryptographic support, E-Commerce protocols, personal information manager (PIM) services, and advanced Internet and telephony services.

6. Alchemy™ includes Rapid Application Development (RAD) objects that leverage available Alchemy™ services into functionally complete application implementations. RAD objects can be customized to rapidly deliver applications with a device-specific look and feel.
7. Alchemy™ includes a comprehensive set of development tools (e.g., installation tools, system configuration tools, debugging and testing tools) that smooth the development process.

The remaining sections of this paper describe the architecture and capabilities that Alchemy™ delivers.

2 Architectural Overview

The basic architecture of Alchemy™, as deployed on Touchstone™, is shown in Figure 1.

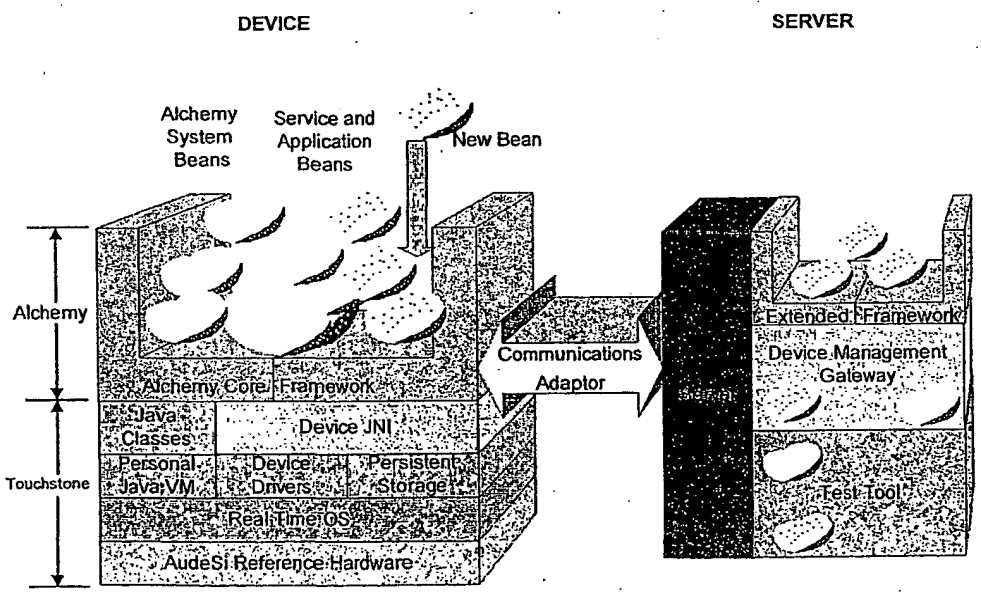


Figure 1 – Alchemy™ Architecture

The major components in the device architecture are described below.

1. *Touchstone™* consists of the reference hardware, the RTOS, device driver suite, the PJava environment and the device driver JNI. Touchstone™ currently supports two commercial RTOS/Java environments, namely Sun's JavaOS4C™, and WindRiver Systems' JWorks™.
2. *Alchemy™ Core Framework:* The framework is the backbone of the software architecture, providing a dynamically configurable multi-application framework and managed bean repository. Managed beans represent the active components in the system, and are directly accessible through the framework. The framework is also responsible for low-level device infrastructure including system configuration, device initialization, communications adapters, memory management, file system, persistent storage, and JAR management.
3. *Alchemy™ System Beans:* A variety of system beans are provided with Alchemy. These beans give other application beans access to system services such as hardware devices, persistent storage, system properties and application support facilities.
4. *Service Beans*: General purpose and application-specific capabilities are provided to the system through service beans. They are device-independent and reusable in nature. Application beans make use of service beans to deliver device-specific application capabilities. While Alchemy™ supplies a number of service beans, they are most often associated with specific application functionality (e.g., the protocol component of an application).
5. *Application Beans*: The look and feel of a specific application on a specific device is implemented in an application bean. An application bean implements the user interface and interacts with system and service beans to achieve the application's intended functionality. Alchemy™ provides a number of RAD objects that are designed to deliver functional applications quickly. RAD objects can be extended or modified to deliver device-specific application features.
6. *Communications Adapter*: The communications adapter allows for sever-based communication with the framework and all managed beans that are registered with the framework. Adapters provide a mechanism whereby managed beans within the framework can be manipulated remotely through a stub or client bean. Alchemy™ currently supports HTTP and HTTPS communications adapters.

The Alchemy™ architecture provides direct seamless integration with server applications through the framework and communications adapters. While the device/server architecture is general-purpose in nature, it is currently used within the Alchemy™ environment to deliver three key capabilities.

1. *Device Management Gateway*: The gateway is part of the Alchemy™ architecture that supports remote management of devices. Through a communications adapter, the gateway can interact with the framework to provide a number of capabilities, including discovery services, device configuration, and application download. Discovery services are used to determine the existing configuration of a device, including available resources, existing software components, and version numbers. Device configuration services allow for remote configuration of the services and applications on the device. Application download services allow for upgrades to the device software load, including download of new applications to the device. The gateway is fully extensible and can be easily incorporated into any server environment.
2. *Remote Testing*: Because Alchemy™ supports remote interaction with the framework and managed beans within the framework, it is well suited to remote device testing. Alchemy™ includes a flexible and extensible observation and control architecture that allows a server-based test tool to monitor and control the behavior of managed beans within the framework. The observation and control architecture is well suited to development time testing, automated regression testing, and in-field testing.
3. *Extended Framework*: Alchemy™ directly supports the concept of an extended framework where a slave framework environment is created on the server and controlled by the device. Using this mechanism, it is possible to partition applications in a manner that places most of the application resources on the server with only the shell of the application deployed on the device itself. Interaction between the device-resident application shell and the server-resident application services is achieved transparently through the framework and the communications adapter.

3 Alchemy™ Core Framework

The framework is the heart of the Alchemy™ architecture and is responsible for establishing and maintaining all basic system services, and orchestrating the device initialization sequence. The services available in the framework are described in the following subsections.

3.1 System Configuration

The framework defines the system configuration through a set of system properties. These properties define precisely the underlying capabilities of the device and include information regarding available memory, display capabilities, and the existence of I/O devices such as Ethernet, USB, modem, smart card reader, etc. In essence, the system properties define a subset of Touchstone™ capabilities that are available to a device-specific software load. System properties can be defined through a combination of hard-coded, command-line, and property-file entries. After initialization, system properties are made generally available through the standard Java system properties.

The system properties can be accessed remotely through discovery services in the gateway.

3.2 Persistent Storage

Alchemy™ relies on the existence of some form of persistent storage. At a minimum, persistent storage is required to hold the software load itself. To take advantage of the dynamic capabilities of Alchemy, writeable persistent storage is required. Beyond storage of the system itself, the framework controls all other persistent storage and makes it available to managed beans within the system. A variety of flavors of persistent memory are supported including NAND Flash, NOR Flash, and EEPROM. Touchstone™ provides underlying hardware and driver support for all types of persistent memory.

Through a set of configuration parameters, the developer can define any number of contiguous blocks of persistent storage within the systems available persistent memory. Once defined, these persistent storage blocks are available for allocation to managed beans through the framework. The framework is responsible for maintaining a persistent map of persistent storage, and returning the blocks to their respective owners at start-up. The framework is also responsible for de-allocation of persistent storage and defragmentation of persistent memory.

Although Alchemy™ does not rely on the existence of a Flash File System (FFS), it does support one. If a device is configured with a FFS, it will exist in a contiguous block of Flash and be made available through the standard Java file system support. The FFS can co-exist with the block-based persistent storage mechanism provided by the framework.

3.3 JAR Repository

The Java software load for a system is maintained in Java Archive Storage (JAR) files in persistent storage. The JAR Repository is responsible for managing the JARs and adjusting the JVM CLASSPATH to include all available JARs. JARs are separated into three categories as follows:

1. *Java System*: The Java System JARs include all the base Java classes the virtual machine (VM) expects.
2. *Alchemy™System*: The Alchemy™ System JARs contain the entire core Alchemy™ classes used in the system including Alchemy™ utility, system, and service classes.
3. *Application*: Application JARs contain the device specific application and service classes for a system.

The CLASSPATH is dynamically constructed with Java System JARs first, Alchemy™ System JARs next, and Application JARs last. Within each subset of the CLASSPATH, JARs are maintained in reverse order of their registration (e.g., last registered JAR is first). In this way new JARs can replace obsolete classes in older JARs. The JAR Repository can be managed remotely through the gateway to add and remove JARs from the system. The JAR Repository is not dependent on a file system, and all JAR downloads are "guaranteed".

3.4 Managed Bean Repository

Managed beans represent the active components in the system, and collectively, implement the capabilities of the device. The framework is responsible for maintaining a persistent repository of managed beans, and making those beans generally available in the system. The following information is associated with each managed bean in the system.

1. *Name*: The name used to reference the bean.
2. *ID*: A unique identifier for the bean.
3. *Class*: The fully qualified class name for the bean.
4. *Type*: The type of bean.
5. *Persistent Storage Block List*: A list that identifies the persistent storage blocks belonging to the bean.

When a bean is registered with a repository, it must have a unique name. In turn, the bean is assigned a permanent unique identifier. A reference to any bean in the repository can be obtained through the framework using either the name or the ID of the bean.

3.5 Communications Adapters

The framework is responsible for establishing any communications adapters that exist in the system. Configuration parameters are used to define which adapter is used. Alchemy™ supports HTTP and HTTPS adapters. If an adapter is not included in the device configuration, remote services will be unavailable.

3.6 System Initialization

The framework is responsible for performing system initialization. The primary goal of the initialization sequence is to bring the device to life quickly and then complete system initialization in the background. The following phases make up the initialization sequence:

1. The framework establishes its environment including system properties, framework, persistent storage maps, and managed bean repository.
2. The framework establishes application services through creation of the application manager. The application manager is discussed in section 4.1.
3. The framework creates and registers all application type managed beans. At registration time, managed beans are given the opportunity to perform required internal initialization. While not enforceable, applications should avoid extensive initialization work at registration time. Wherever possible, object creation and initialization within an application should be deferred to the moment that such objects are required.
4. Once all applications are registered, control is transferred from the framework to the application selector where the default idle application is started. Idle applications are discussed in section 4.1.
5. In the background, the framework continues to create and register other managed beans in the system until it has exhausted the list contained in the managed bean repository. Should a running application require access to a bean that has not yet been initialized, that bean will be created and registered at the time of the request.

3.7 Idle Detection

Idleness of a device can be used to trigger a number of activities ranging from activation of a screen saver, to performing systems maintenance activities such as defragmentation of persistent storage. Alchemy™ provides an idle management service that controls the idle detection process and dispatches events indicating system idleness. Two mechanisms are provided for indicating activity in the system.

1. *Spontaneous*: The idle manager provides a direct API used to indicate current activity. Any object in the system can indicate current activity by making this call.

2. *Polling*: Objects in the system can act as polled activity sources by registering with the idle manager. The manager uses a configurable idle timer to set the polling period. On each timer expiration the manager polls each source for activity status.

On each idle timer expiration, the manager decides if the device has been idle during that period. If it has, the manager generates events to all idle listeners. Idle events include an idle count indicating the current number of consecutive idle periods. Listeners can use this count to distinguish between varying levels of idleness. The idle management architecture is illustrated in Figure 2.

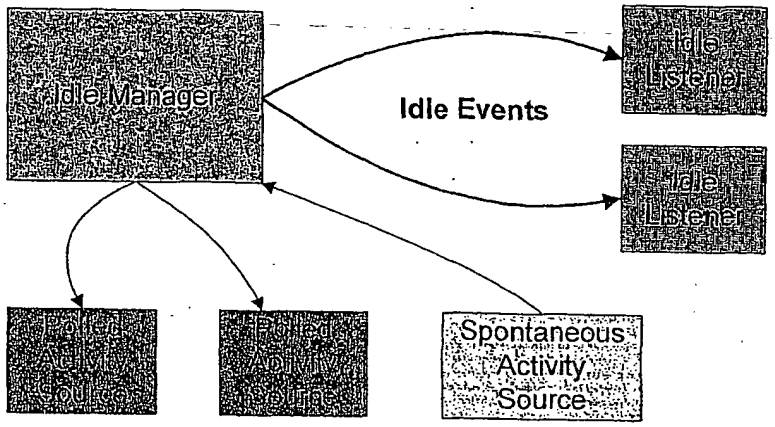


Figure 2 – Idle Management Architecture

4 Application Services

One of Alchemy's primary goals is to provide an environment that allows the developer to focus on application-level development. Towards this goal, Alchemy™ provides an extensible multi-application architecture in which applications can co-exist. In addition, Alchemy™ provides a number of common application services and RAD objects that further reduce the development cycle for applications.

By definition, an application must include some user interface component (a means by which the user interacts with the application). Unlike traditional computing environments, it is not a certainty that an intelligent device will include a full-featured, point-and-click windowing environment in which the user interface (UI) can be deployed. Alchemy™ provides the necessary infrastructure to support the following display environments.

1. *Abstract Windowing Toolkit (AWT) Windowing Environment*: This is the standard Java GUI environment that supports graphics, windows, and a pointing device. Standard AWT-based drawing components and event mechanisms are applicable to this environment.
2. *Sea-of-Dots Graphics Environment*: This type of display supports rudimentary bit-mapped graphics, icons, and text. A pointing device is not normally associated with this environment. Configurable soft keys may be associated with this environment.
3. *Alphanumeric Display Environment*: This type of display supports some number of lines and columns of alphanumeric style text and fixed-size icons. Configurable soft keys may be associated with this environment.

Alchemy's application services are organized into display-independent and display-dependent capabilities. Display-independent services include the multi-application architecture, options management, and language management, which are described in section 4.1 through 4.3. Display-dependent services for the three display environments are described after the display-independent services.

NOTE: The first release of Alchemy™ includes only AWT-dependent display services. Sea-of-dots and alphanumeric display services are deferred to a subsequent release of Alchemy.

4.1 Multi-Application Architecture

The multi-application architecture in Alchemy™ is supported through an Application Manager that provides three basic services as illustrated in Figure 3 and described below:

1. *Application Registration*: Before applications are available for selection they must be registered with the Application Manager. The registry maintains a reference to each application and can activate and deactivate applications on demand. Within the registry, a single application is identified as the idle application, which is activated automatically at start-up, and whenever the device becomes idle. The registry also generates events indicating when the set of applications is modified. This event mechanism supports dynamically modifying the set of available applications.
2. *Application Selection*: Applications are given focus and made active by the user through an application selector. The application selector interacts with the registry to present the user with a selection of available applications and effects the activation of applications when they are selected. By definition, an application selector includes some UI component, and as such must be implemented for a specific display environment. Alchemy™ provides both display-independent services for application selectors as well as display-specific RAD implementations of application selectors.
3. *Application History*: A history is maintained when applications are activated and de-activated. When an active application is deactivated, the application history determines which application should be activated. Alchemy™ provides the base services for application history, as well as a RADO implementation of a configurable, fixed-depth, LIFO history.

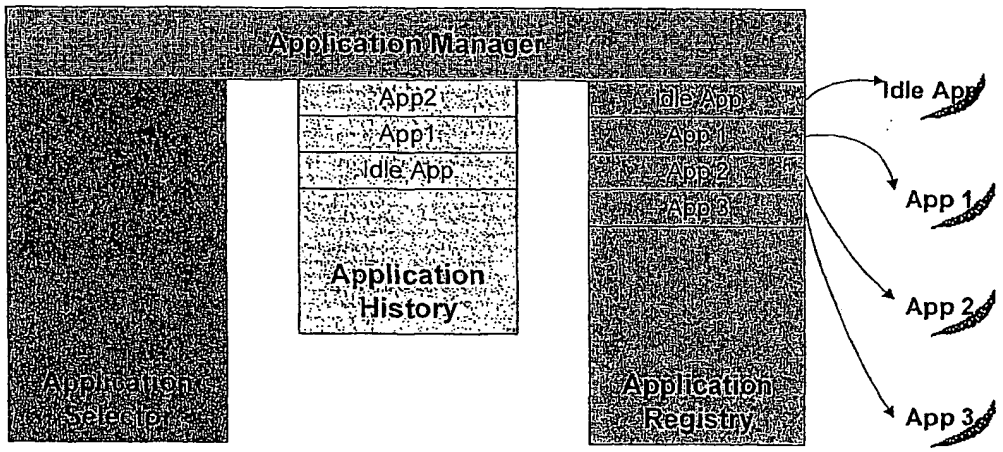


Figure 3 – Multi-Application Architecture

4.2 Options Management

Most device environments have a requirement for persistent, user settable options. These options range from global options that apply across the entire device (e.g., language preference), to application or service specific options. While the scope of options may vary, user access to options is typically available through a common access point. In a dynamic application environment it becomes necessary for options management to become dynamic as well. Alchemy™ provides an extensible options management service that facilitates the organization of options, without imposing a specific implementation of options management at the UI level. The options service in Alchemy™ provides a base infrastructure that includes

an options menu object capable of supporting a hierarchically representation of options and submenus of options, and an abstract option screen object from which an actual options editing implementation can be derived. Furthermore, the options manager contains a mechanism for identifying start-up options that must be set on initial start-up or each time the device is activated. The options management architecture is illustrated in Figure 4.

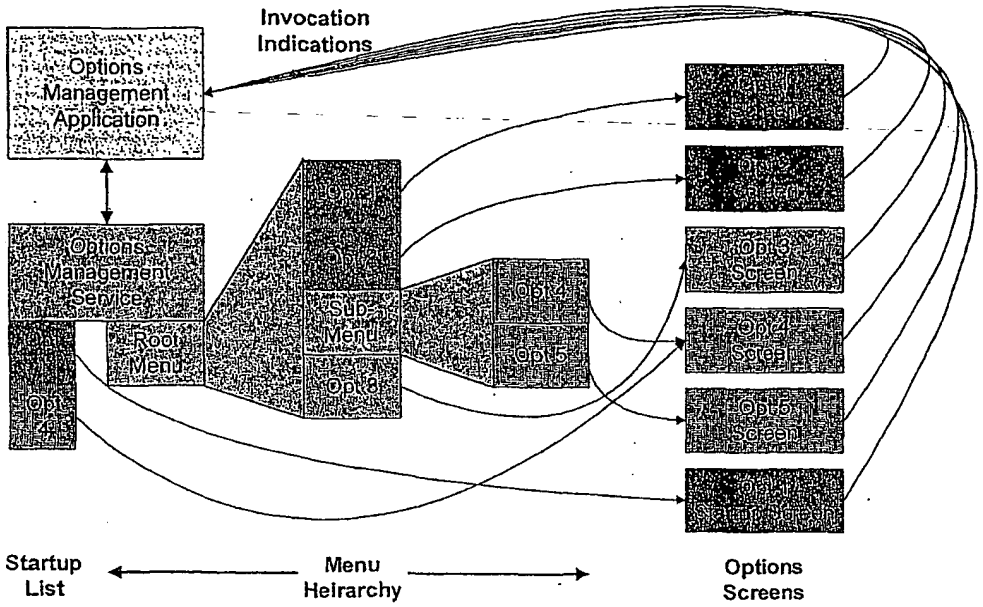


Figure 4 – Options Management Object Relationship

Key elements of the options architecture are as follows:

1. *Options Management Application*: This is the application responsible for presenting the common access points to options.
2. *Options Management Service*: This Alchemy™ service provides a root menu and a start-up list for the system.
3. *Root Menu*: This is the root of the options tree. Options and submenus can be added, as required.
4. *Startup List*: This is the list of options that need to be initialized at device start-up. These options must be set on initial start-up or each time the device is activated.
5. *Options Screens*: These are the actual GUI implementations of options editing screens for the available options.
6. *Invocation Indications*: Alchemy™ decouples the invocation of an options screen from the user's dismissal of the screen. When the option screen is dismissed, the invocation indication is used to inform the controlling application that user manipulation of the options screen is complete.

4.3 Language Management

Alchemy™ provides a language management service that maintains a list of available languages and a listener interface that supports notification of language changes. In addition, Alchemy™ provides a base language interface and a core language implementation from which application specific language object

can be derived. The language architecture is illustrated in Figure 5 for an application that extends the Alchemy™ core language interface.

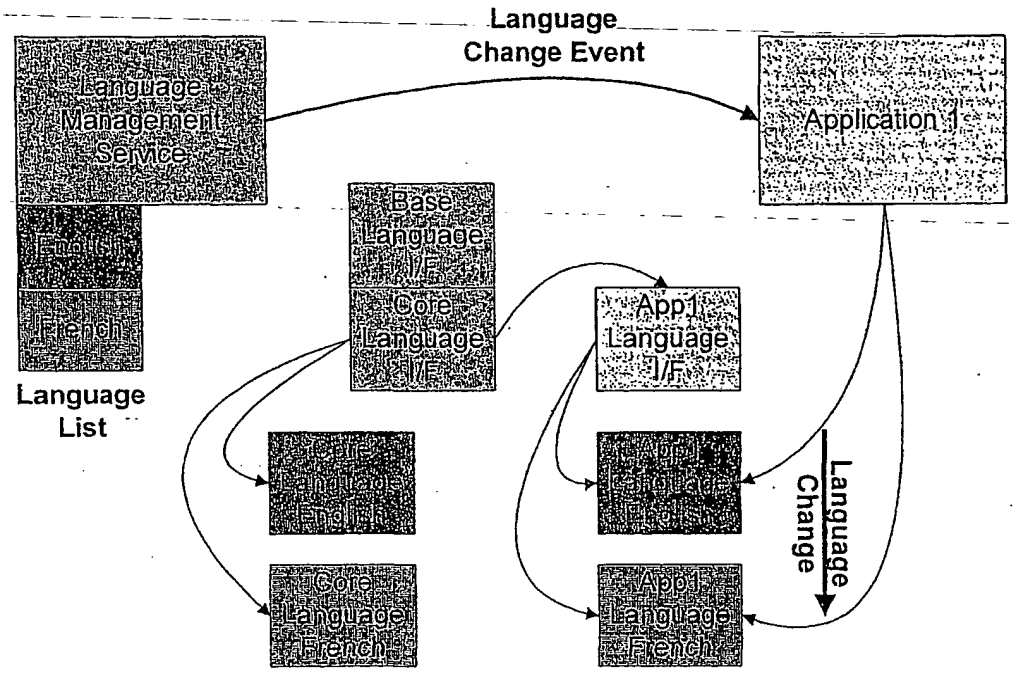


Figure 5 – Class Structure Language Management

4.4 AWT Services

In an AWT-style application environment, the developer has access to all AWT services supplied with PJava. While AWT services form the basis of the GUI application, Alchemy™ provides a number of services, interfaces, and objects that tightly couple the AWT-based application to the multi-application architecture.

4.4.1 Applications, Icons, and Selection

AWT applications extend the base application interface with two objects, a generic panel, and an Alchemy™ specific AWT icon. The panel is used to house the GUI for the application itself. Whenever the application is activated, the panel is displayed. The icon is used to effect application selection. It contains an image associated with the application and a reference to the application. The icon generates "application selected" events when a mouse click occurs on it. The sequence involved in selection of an application is described below and illustrated in Figure 6.

1. A mouse click in the application icon generates an "application selected event" which is received by the application selector.
2. The application selector makes a request to the application manager to activate the application. A reference to the application is supplied from the application icon.
3. The application manager informs the application to activate and requests a reference to the application display panel.
4. The application manager displays the panel and control is turned over to the application.

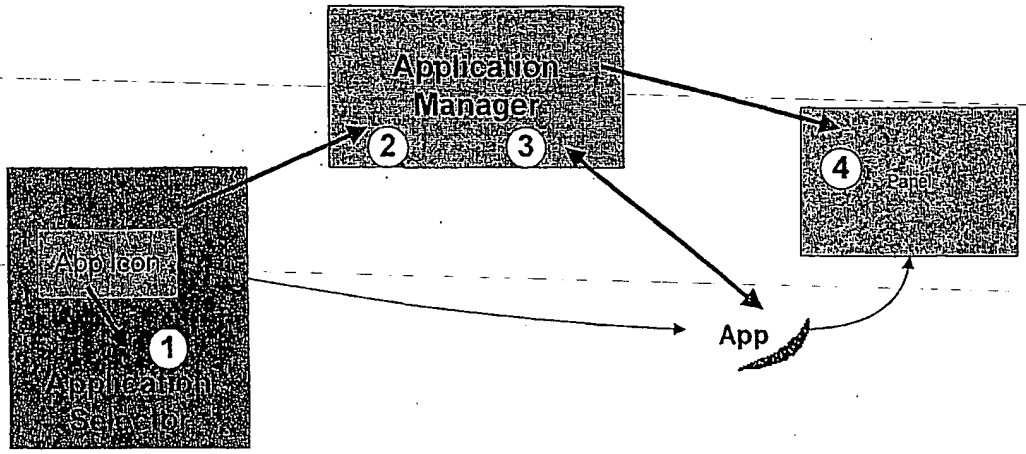


Figure 6 – Application Selection

To speed development, Alchemy™ provides an AWT application RAD object that implements a shell application. This RAD object can be extended to achieve specific application functionality.

4.4.2 Application Selectors

Alchemy™ provides an abstract base class for AWT application selectors. Several RAD AWT application selectors are also included with Alchemy™.

4.4.2.1 Sidebar Selector

The AWT Sidebar Application selector implements an icon strip along one boarder of the screen that contains a set of icons associated with the device. When an icon is selected, its associated application is selected and displayed on the remainder of the screen. Scroll bars become active when the number of application icons exceeds the available real estate available to the sidebar. The sidebar application selector is illustrated in Figure 7.

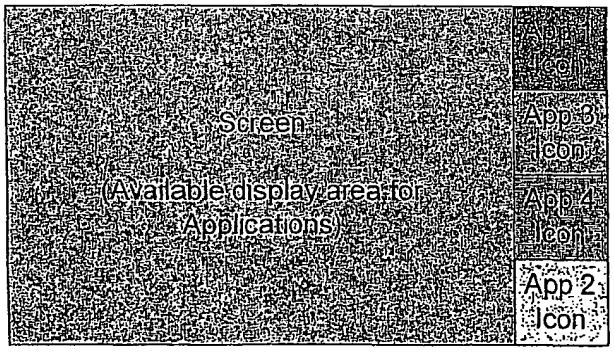


Figure 7 – Sidebar Application Selector

4.4.2.2 Corner Selector

The AWT Corner Application selector implements a small icon in the corner of the screen that expands into a grid of application icons. When an icon is selected, its associated application is selected and the selector minimizes in the corner. The corner application selector is illustrated in Figure 8.

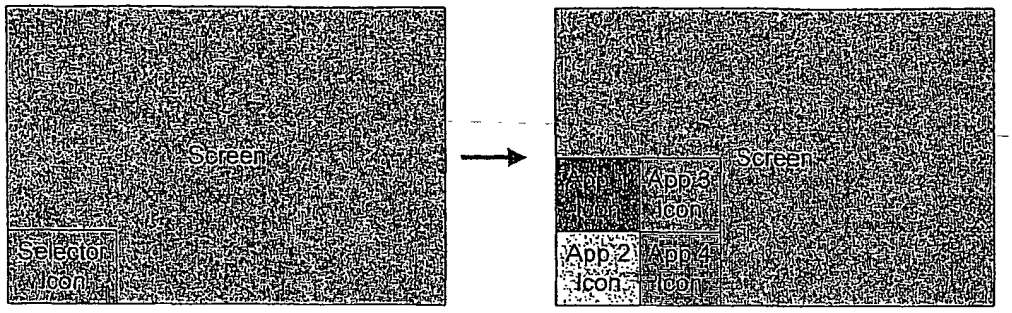


Figure 8 – Corner Application Selector

4.4.3 Options Manager

Alchemy™ provides a RAD object that implements a pull-down-menu-based options management application. It dynamically maintains the nested options associated with a device, and provides access to those options through a nested pull-down menu. The options manager supports options screens that are implemented as panels.

4.4.4 Virtual Keyboard

Because many devices will be touchscreen-based, Alchemy™ provides full virtual keyboard support including an image-based, configurable virtual keyboard, and a virtual keyboard manager that provides tight integration between the virtual keyboard and text fields within an application GUI.

The virtual keyboard is based on a multi-image architecture where a visible image contains the user's view of the keyboard, and an underlying invisible image provides a mapping to key values. Mouse clicks on the visible image are mapped by location (X, Y) onto the invisible image and the corresponding value in the invisible image is converted into a key value. The virtual keyboard generates all standard AWT key events. The virtual keyboard also supports automated image replacement based on key presses. For example, when the shift key is pressed, the keyboard image can be changed to indicate capital letters.

When active in a system, the virtual keyboard is automatically displayed when a text field gains focus. The text field is reproduced on the keyboard and input is accepted from the user. After completion of the entry, the keyboard disappears and the text field is updated to contain the appropriate text. While this mechanism is automatic and requires no special consideration from the GUI developer to function, it can be cumbersome within a GUI implementation. First of all, the virtual keyboard covers the text that is being edited, so auxiliary information associated with the field (e.g., a field description) is likely to be hidden. Secondly, there is no general-purpose mechanism available to identify a particular keyboard style to associate with a particular text field. Lastly, moving from field-to-field involves dismissing the keyboard when one field is completed and re-displaying it when another field is given focus. Alchemy™ overcomes these problems by providing a virtual keyboard manager.

The virtual keyboard manager tightly couples a set of text fields in a GUI implementation to the virtual keyboard by providing the following capabilities.

1. A field descriptor can be associated with a text field that is registered with the manager. This description will be displayed on the keyboard while the text is being edited.

2. A specific style of keyboard can be associated with a text field that is registered with the manager.
3. "Tab-ordering" can be established for a group of text-fields, where the keyboard can be used to navigate from field-to-field without being dismissed and redisplayed.

An application environment that leverages the virtual keyboard manager can be made more user-friendly than one that does not.

Alchemy™ comes with a number of RAD virtual keyboards including standard qwerty, alphanumeric, and keypad styles.

4.4.5 Activity Source

In support of idle detection mechanism, Alchemy™ provides a polled activity source for AWT environments. This source monitors AWT mouse and key events, and provides an indication of activity to the idle manager when polled. This source eliminates the need for individual applications to continually indicate activity to the idle manager (based on the premise that if the user is interacting with the device through the mouse or keyboard, then some application on the device must be active).

5 Smart Card Services

An implementation of the Open Card Framework (OCF) is the cornerstone of Alchemy's smart card services. OCF is implemented within Alchemy™ as a service bean and includes terminal and terminal factory implementations for the ARP smart card reader, as well as a number of 3rd party readers including the Gemplus GCR410 and Litronic 210. Using OCF, it is possible to support any smart card application by implementing an OCF service for the card application, and a terminal application that uses the service. Alchemy™ includes a number of RADOs that act as example service and terminal application implementations.

Because Alchemy™ is a dynamic multi-application environment, it is designed to support multi-application card environments. To begin with, Alchemy™ provides a card detection service capable of asynchronously detecting card insertions and removals, and notifying registered listeners. When a card is inserted, existing OCF services are leveraged to determine the available card applications and a list of those applications is returned to the listener. Based on this list, an application selector or some other service within the device can select and activate a particular terminal/card application pair.

Alchemy™ also supports dynamic download of smart card applications including both terminal and card resident portions of a smart card application. Download support is delivered through the gateway as described in section 8.4. Specific OCF services are available in Alchemy™ which support the actual download of applications to the card.

NOTE: The first release of Alchemy™ includes support for the Java Card environment only. Support for MultOS and other multi-application environments is deferred to a subsequent release of Alchemy™.

Finally, the OCF service is augmented with observation and control mechanisms that support remote monitoring of ISO 7816 messaging at the card terminal. This capability facilitates debugging and testing of smart card applications. The observation and control architecture is described in section 8.1.

5.1 Java Card

Alchemy™ provides direct support for the Java Card 2.0 and 2.1 environments including a full implementation of the Java Card simulation environment. Simulation support is realized through the Java Card simulation manager, which allows the developer to create any number of simulated terminals and any number of simulated Java Cards. Each simulated card can be configured to contain a number of real Java Card Applets. The manager controls the insertion and removal of a simulated card into a simulated terminal. Once card insertion occurs, the interaction between card applets and card terminal applications can occur just as in a real Java Card environment. Using the simulated environment, it is possible to develop a card applet and terminal application in the workstation environment and then migrate it to an actual terminal/card environment. Alchemy's Java Card environment is illustrated in Figure 9.

The Java Card simulation manager is augmented with observation and control mechanisms that support remote control of card insertion and removal. See section 8.2 for details.

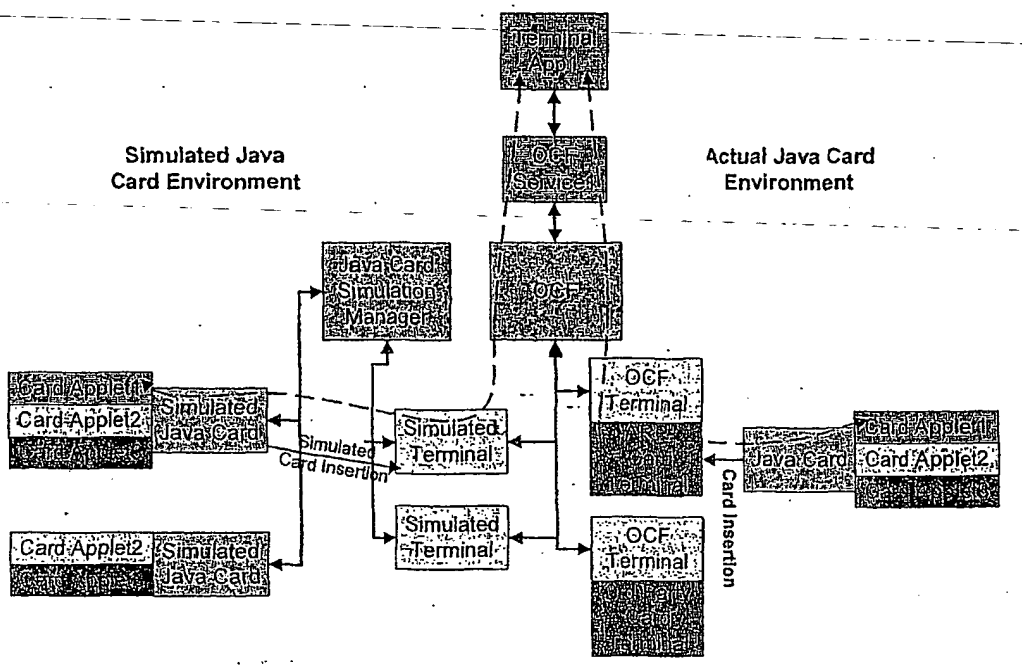


Figure 9 – Java Card Environment

Alchemy's implementation of OCF contains specific services that support download and deletion of applets on a Java Card. This support is leveraged by the gateway to achieve remote management of the application suite on a Java Card. See section 8.4 for details.

6 Telephony Services

Alchemy™ provides a number of telephony services including modem support, basic telephony support, and advanced telephony support. These services are described in the following subsections.

6.1 Modem Services

Modem services are based on Javax.comm, which defines a standard mechanism for serial communications in Java. The modem service extends the Javax.comm serial I/O API with a modem control API that provides the functionality to initialize and configure the modem. The existing implementation is targeted at the V.90 modem available on Touchstone™, but the modem service is easily portable to other modem solutions.

6.2 Basic Telephony Services

The basic telephony services available in Alchemy™ are closely coupled to the Touchstone™ Telephony board capabilities. The basic telephony service bean provides a control API for direct control over telephony features, a status API for retrieving telephony status, and an event listener API for reception of asynchronous telephony events. All basic telephony services are summarized in the following table.

Table 1: Basic Telephony Services

Feature/Description	Control	Status	Event
Initialization/Register Control	X		
Volume			
Handset	X	X	
Speaker	X	X	
CAS Detection			X
FSK Data Reception			X
Type I Caller ID			X
Type II Caller ID			X
Visual Message Waiting			X
DTMF Tone Generation	X		
Ring Detection			X
Extension In Use Detection		X	X
Distinct Audible Ring Tones	X		
Voice Path Control			
Handset	X	X	
Handsfree	X	X	
Microphone	X	X	
Hold/Mute/Unmute	X	X	
Dial Pad Scan			X
Cradle Hook switch Scan		X	X
Hook Switch Control			
On/Off hook	X	X	X
Flash	X		X

6.3 *Advanced Telephony Services*

While the basic telephony services described in the previous subsection provide access to all the telephony capabilities of the ARP, and as such, give the developer fine-grained control over these capabilities, the onus is on the developer to integrate these capabilities into a functional telephony service. Alchemy™ includes a number of advanced telephony services that provide this higher level of integration, thus reducing the development effort required to deploy a telephony application. Advanced telephony service include the following:

1. *Voice Call State Machine*: Provides intelligent control of incoming and outgoing voice paths with control over handset/handsfree modes, hold, and mute. Also includes persistent volume control for both handset and handsfree, and persistent audible ring tone selection.
2. *Hook Switch State Machine*: Provides integrated control and feedback of the hook switch and hook switch cradle.

3. *Telephony/Modem Line Sharing State Machine*: Provides intelligent sharing of the line between telephony and modem functionality in a single-line environment.
4. *Distinctive Ring Interpretation*: Provided the ability to define distinctive ring interpreters that can be used in conjunction with CLID and other services to uniquely identify incoming calls.
5. *Audio Streaming and Playback*: Provides voice path control and decompression streams for the audio protocols supported on the ARP.
6. *Contact Database*: Provides a configurable contact database that tightly integrates with CLID features. Provides underlying capabilities for caller log, preferred name matching, area code striping, etc.

Alchemy includes a set of RAD objects that implement a full-featured telephony application based on the advanced telephony features described above. This application serves as both an example, and a starting point for a device specific telephony application.

7 Internet Services

Alchemy™ provides seamless access to the Internet, and as such, is designed to develop "Internet Aware" devices. Specifically, Alchemy™ provides connection services, integration of 3rd party Internet applications such as browsers, and email clients, and custom URL handling for services and applications that require integration with browser services.

7.1 PPP and Auto Dial

Dialup connectivity to the Internet via an ISP is supported through Alchemy's Point-to-Point Protocol (PPP) and Auto Dial service. The PPP and Auto Dial service maintains a persistent list of ISP configurations. Each configuration contains the information required to make an ISP connection, including phone number, user ID, password, DNS Addresses, mail server, etc. The PPP and Auto Dial service provides an API that supports addition, deletion and selection of ISP configurations. Once selected, a particular configuration is used to automatically establish a connection when some other service or application activates TCP/IP services. The PPP and Auto Dial service is also remotely controllable through observation and control services.

7.2 Internet Applications

Alchemy™ does not reinvent the wheel when it comes to Internet applications such as browsers and email clients. A number of Java-based "embeddable" Internet application suites are commercially available from 3rd party vendors, and these application suites can typically be integrated into Alchemy™ with little effort. Alchemy™ has been demonstrated to work with two Internet applications suites, namely Sun Microsystems' Personal Applications™ suite, and Espial Group's Escape™ browser and Ebox™ email client. While Alchemy™ provides integration for these application suites, the suites must currently be obtained directly from their respective vendors.

7.3 Custom URL Handling

The existence of a browser on a device is useful for more than just traditional Internet browsing applications. The HTML rendering capabilities of the browser can be leveraged into other on-device applications to deliver UI services. In such an environment, the developer can use custom URL definitions to trigger events in other services or applications. Alchemy™ provides a custom URL dispatching service capable of maintaining a list of custom URL types, and a set of listeners for each type. When the browser encounters an unknown URL type, it hands it to the dispatcher which, in turn, dispatches the URL to all interested listeners.

This process is illustrated in Figure 10.

As an example, consider the following:

Alchemy:saveToDirectory?name=AudeSi Technologies Inc.&number=403-730-7555

This URL is not recognized by the browser so it is passed to the Custom URL dispatcher. It is then passed to the URL listeners. One of the listeners is the directory application that takes the URL and adds a new entry for AudeSi Technologies Inc.

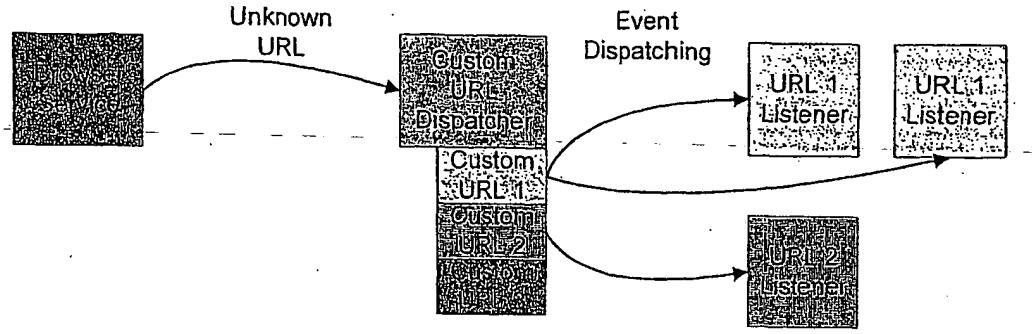


Figure 10 – Custom URL Dispatching

8 Observation and Control Services

The distributed nature of the Alchemy™ core provides direct support for remote instantiation of objects into a device load. Leveraging this mechanism, Alchemy™ delivers a complete observation and control (O&C) architecture capable of supporting system debugging, remote diagnostics, automated regression testing, and device management.

8.1 Observation and Control Architecture

The O&C architecture delivers two key capabilities to the developer.

1. *Observation*: The ability to extract information from the system is accomplished by implementing a specific observable interface within an object, and remotely controlling the observation process through an observer bean. Actual observations are passed from the device through a remote event mechanism. Observations can be generic (e.g., text-based logging) or specific (e.g., ISO 7816 messaging).
2. *Control*: The ability to remotely control objects in the system is accomplished by implementing a specific controllable interface within an object and remotely controlling that object through a controller bean. Control functions are typically object specific in nature.

Alchemy's™ O&C architecture and its major components are described below and illustrated in Figure 11.

1. *O&C Registry*: On the device side, the O&C registry is responsible for maintaining a list of all observable/controllable objects in the system. The registry communicates with a remote O&C manager to orchestrate a specific O&C session. All observable/controllable objects in the system must register with the registry to be actively observable/controllable.
2. *Observable/Controllable Object*: Any object in the system is observable/controllable if it implements a specific observable/controllable interface, and it registers itself with the O&C registry. The object is manipulated through a mated observer/controller object that knows the specific O&C interface implemented by the object.
3. *Observer/Controller Managed Bean*: On the device side, the Observer/Controller is instantiated as a temporary managed bean, and is responsible for manipulating observable/controllable objects of a specific type. The observer/controller is remotely controlled by the O&C manager through a matching client bean on the server side.

4. *Observer/Controller Client Bean*: On the server side, the O&C manager transparently communicates with the device side observer/controller through the observer/controller client bean.
5. *O&C Manager*: The O&C Manager controls the observation process through three primary functions.
   I. The manager maintains a registry of observer/controllers that can be activated in the framework. When instructed to do so by some controlling test tool, the manager instantiates and activates an observer/controller in the framework.
   II. The manager presents a configuration and control interface for test tools. It identifies all available observer/controllers, and allows the test tool to configure, activate and deactivate those objects.
   III. The manager processes observation events and submits them to observation database.
6. *Observation Database*: The Observation Database stores all the observations for a session. Each observation is stored with a time-stamp, an observation class name, and the actual observation object.
7. *Test Tool*: Any controlling application can act as a test tool. A test tool may present a GUI to the user that allows him to configure, activate and deactivate observer/controllers in the system, or it may be a self-contained automatic test-harness. Test tools interact with the O&C manager to control an O&C session, and with the observation database to inspect observations.

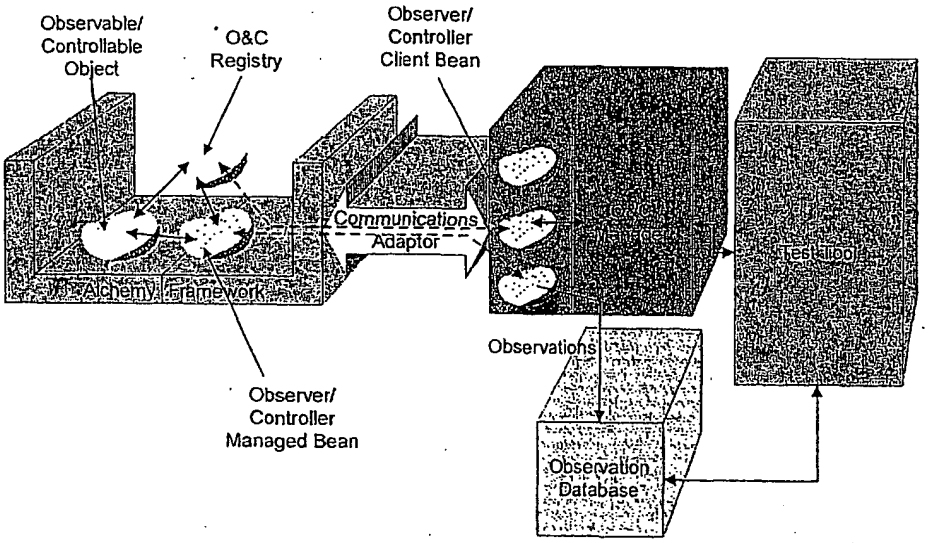


Figure 11 – Observation and Control Architecture

Because the O&C architecture is open and extensible, it can be leveraged to deliver any specific O&C capabilities applicable to a particular device or service. Alchemy™ provides specific O&C capabilities through a number of tools, and observable/controllable objects as described in the following subsections.

8.2 Alchemy™ Observer/Controllers

Alchemy™ provides a number of specific observer/controller objects that can be used directly by the developer including the following:

1. *Text-based Logger*: This observer performs generic text-based logging of events, and can be used for general-purpose tracing.

2. *Application Selection Observer*: This observer generates observations that indicate application selection and de-selection activities.
3. *ISO 7816 Observer*: This observer generates observations for all smart card messaging that occurs between an OCF terminal and a smart card.
4. *Java Card Simulation Observer/Controller*: This observer/controller interacts with the Java Card Simulation Manager to identify available simulated Java Cards and facilitates simulated insertion and removal of those cards into a simulated terminal.
5. *Basic Telephony Observer*: This observer handles all basic telephony observations generated by the Alchemy™ basic telephony service.
6. *Persistent Storage Observer*: This observer generates observations that contain persistent storage block maps.
7. *Modem Observer*: This observer generates observations that include all control commands sent to the modem.
8. *PPP and Auto Dial Observer*: This observer generates observations that identify all activities associated with PPP connections.

8.3 Alchemy Probe

Alchemy™ includes a developers' probing tool. This tool provides a controlling GUI that is used to establish and control an O&C session. The probing tool was developed to support testing of Alchemy™, and as such, works with all Alchemy™ supplied observer/controllers. The probing tool is based on an extensible architecture and can be easily modified to support any custom observer/controller. The main features are described as follows:

1. *Configuration*: Developers can configure an O&C session by establishing a connection to the device under test, and identifying the specific observer/controllers that are active.
2. *Watchpoints*: Developers can set watchpoints for particular observations of interest.
3. *Filtering*: Developers can define filters on the properties of the observations.
4. *Database Interaction*: The probing tool provides complete integration with the observation database for the storage and retrieval of observations. You can also archive and playback previous sessions.
5. *Formatters*: Observations are not necessarily in human-readable form. The probing tool provides the infrastructure to support custom interpreters that can interpret and display specific observations.
6. *Controller Interfaces*: The infrastructure supports custom user interfaces that can be used to activate the control interface of observer/controllers.

8.4 Device Management Gateway

The Device Management Gateway provides remote management facilities for devices under development, as well as deployed devices. The gateway is based on the Alchemy™ O&C architecture and delivers the following key capabilities:

1. *Discovery*: This process involves determining the precise configuration of a device. Specifically the following information can be retrieved from a device:
   I. *System Properties*: System properties can be retrieved either individually or as a group.
   II. *Managed Beans*: The managed bean repository can be queried for the existence of an individual bean or for a list of all existing beans. Managed bean queries return bean name, class, version, and dependency information.
   III. *JARs*: The JAR repository can be queried for a list of JARs and their CLASSPATH ordering. The amount of free space in the repository can also be determined.

IV. *Java Cards*: Card terminals can be queried for the existence of Java Cards, and subsequently, each Java Card can be queried for a list of Java Card applets.

2. *Bean Management*: This process involves manipulating the set of managed beans that exist in the device. Managed beans can be added, replaced, and removed. In effect, this service allows the application and service suite on the device to be customized remotely. Download services include automatic instantiation of new applications, automatic device restart (where required), and robust recovery from failed downloads.

3. *JAR Management*: While bean management services automatically support downloading of new JARs associated with new applications and services, the gateway supports direct manipulation of JARs that exist in the JAR repository. Specifically, the gateway supports reordering of JARs in the JVM CLASSPATH, explicit deletion of JARs, and defragmentation of the JAR repository itself.

4. *Java Card Management:* A Java Card that is inserted into an available card terminal can be managed through the gateway. Specifically, applications can be downloaded to the card or deleted from the card.

The gateway is implemented as an observer/controller managed/client bean pair, and as such, provides gateway capabilities to any server side controlling application. Within Alchemy™, gateway facilities are deployed into the probing tool through a custom control/interpreter interface. This interface acts as an example deployment of the gateway that can be modified for a particular service environment, but also delivers full-featured gateway capabilities to the developer.

9 Development Environment

The underlying goal of Alchemy™ is to reduce the development cycle for intelligent devices by providing a software infrastructure and service suite that can be rapidly deployed onto a hardware reference platform. While application development can be accomplished in any environment that supports Java, ultimately the Java code must be deployed into the target device on top of a properly configured RTOS/driver load. The Alchemy™ development environment supports this process through a set of utilities and tools that augment and enhance your preferred development environment, without imposing a specific environment or tool chain on the developer. Furthermore, all Alchemy™ tools and utilities are written in Java and are delivered with full source code so that they can be tailored specifically to a preferred environment. This means familiar development tools such as Java Integrated Development Environment (IDE), RTOS IDE, and code compactors such as the Embedded Java toolchain can be woven together into a seamless development environment tailored specifically to your needs. While the development environment is adaptable to any hardware target, Alchemy™ provides specific integration for Touchstone™, and will work with it out of the box.

Alchemy™ includes support for the following aspects of the development cycle:

1. *Installation*: Installation wizards and scripts are included that perform a complete installation of the Alchemy™ system into your development environment.

2. *Development*: Alchemy™ includes a complete makefile structure that will build all Alchemy™ software components. This structure can be easily extended to include new applications and services.

3. *Packaging*: Alchemy™ provides packaging tools that perform the following functions:

   I. *System Configuration Definition*: Allows the developer to specify a particular hardware configuration that will be supported for a particular device.

   II. *RTOS/Driver Configuration*: Based on system configuration information, a RTOS/driver configuration can be automatically generated.

   III. *Application Suite Definition*: Allows the developer to define an application load for deployment onto a device. All application/service bean dependencies are automatically resolved to ensure a complete load.

   IV. *Code Compaction and Obfuscation*: Based on a defined application load, code compaction and obfuscation can be performed to minimize the memory footprint of the load.

   V. *JAR Creation*: Single or multiple JAR files can be produced for download to the device.

4. *Target Deployment*: Alchemy™ includes boot-loader facilities for Touchstone™ that are capable of downloading a target software load to the device. Support is available for stand-alone loads as well as network-based loads.

5. *Test/Debug*: As described earlier, Alchemy™ supports a remote observation and control environment for target testing and debugging. A number of test and device management tools are deployed under this architecture, including the probing tool and the gateway.

10 Appendix A – Touchstone™

Touchstone™ is a two-board solution composed of a logic board that contains central processing and the bulk of the I/O capabilities, and a telephony board that includes the telephony and modem capabilities. The functional specification for each of these boards is given below.

Logic Board Functionality

1. *MPC823(E) Processor Socketed*

2. *Core Memory*
   - CS0 boot Flash - 4 Mbytes of onboard Flash, Full 32 bit interface
   - Nand Flash – 4 to 8 Mbytes of onboard, 16 bit interface
   - Compact Flash memory expansion (2-48 Mbytes modules)
   - ATA interface vs. linear Flash will be dependant on existing Java OS4C (i.e., Chorus) support
   - 32Mbytes (max.) dedicated onboard SDRAM operating at the maximum processor bus speed.

3. *MPC82x LCD Controller*
   - *Common* onboard *LCD interface* to provide complete access to all on chip LCD pins or dedicated LCD controller
   - *Dedicated onboard LCD/CRT Controller with 256Kx16 dedicated video DRAM, H/W provision only, EPSON SED1355/6F0A - for part details* www.erd.epson.com

   Target LCD for Reference Board

Manufacturer	Part Number	Pixels	Display Area (H) x (V) mm	Technology
Sharp	LQ64D341	640 x 480	130 x 97	TFT Color

4. *3rd Party Resistive Touch screen*

5. **Keypad, status indicator Interface**
   - Offboard 2mm header to accommodate up to 6 x 6 keyscan matrix
   - I^2C connections to provide 8 incremental GPIOs for LEDs (PCF8574)

6. **16550 UART**
   - Memory mapped UART with H/W flow control and dedicated interrupt
   - Dedicated interface to modem V.90 on telephony board

7. **Communication Ports**

   All Channels use the CPM capabilities and therefore will be constrained only by the performance of the CPM and its Time Division Multiplexing capabilities.

   SCC2
   - IEEE 802.3 compliant (based on MC68160 (Ethernet))
   - IrDA (2.4K to 4Mb/s) - H/W only

   SMC1
   - 1 - 7816 compliant full size, smart card interface

   SMC2
   - Non-isolated RS232 port

   USB
   - Dedicated keyboard interface – non compliant with USB host standards

I2C

- 8 additional onboard GPIOs (PCF8574)
- Digital potentiometer – LCD brightness control
- 16 Kb EEPROM

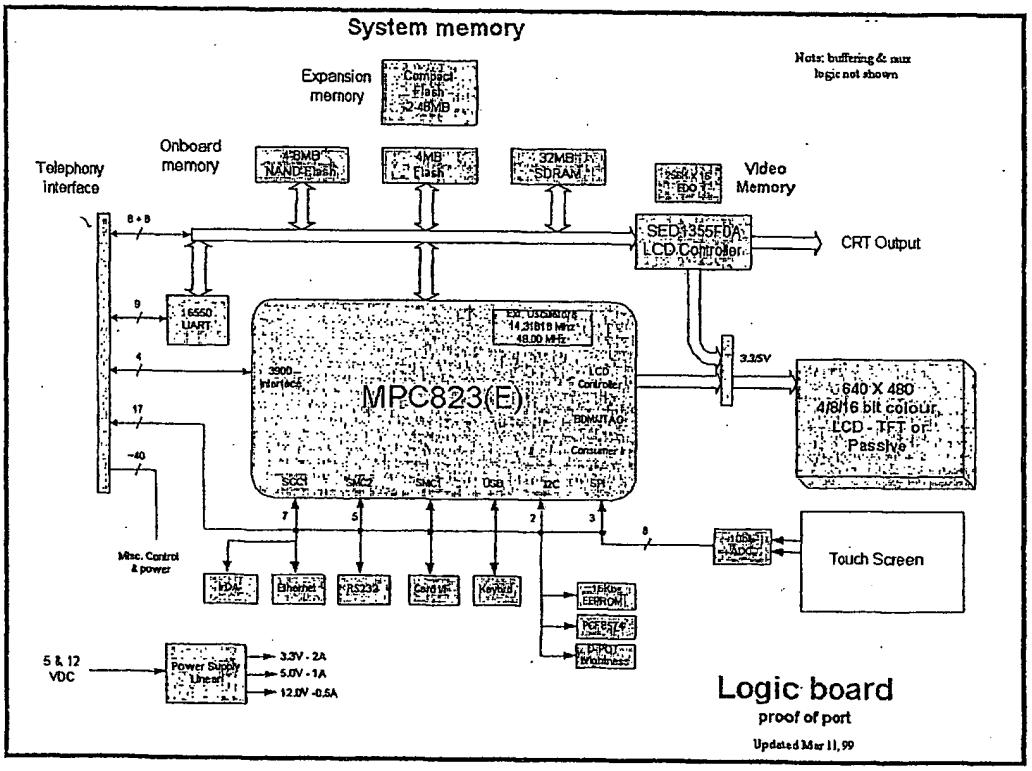
SPI

- Touch screen A/D interface

8. Development and Debug Capabilities

- MPC823 BDM port (non-isolated)
- Dedicated Logic analyzer ports for primary MPC 823 address, data and control logic
- Memory mapped debug LED's 4 character, 5x5 ASCII interface

Logic Block Diagram



Telephony Board Functionality

Processor-Independent Telephony Solution

1-line or 2-line operation – user configurable

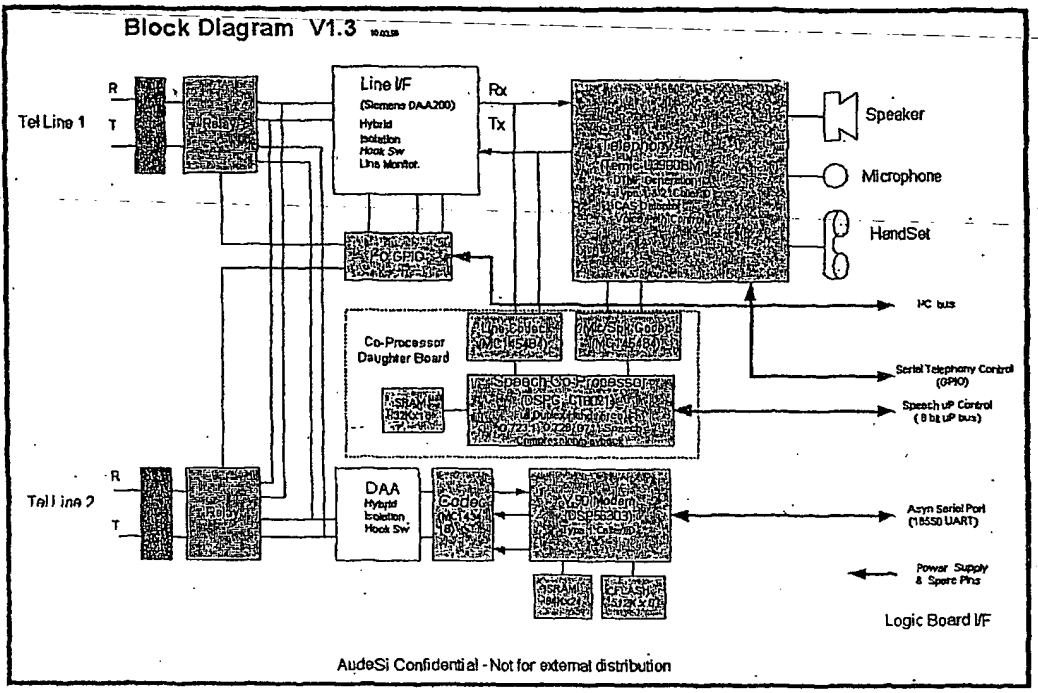
1. **Line One Features**
   - Full featured handset/handsfree capability
   - Type I and II Caller ID features.
   - Full duplex handsfree
   - Shared Voice or modem (1-line scenario)
   - Isolated PSTN interface
   - Does not include DTAD or Voltage Message Waiting capability
   - Powered only operation i.e., no POTS operation

2. *DSP-based Handsfree and TrueSpeech ™Audio (Line 1 only)*
   - Integrated Full Duplex Handsfree
   - DTMF/Tone generation and detection
   - Audio streaming de/compression as per DSP Group CT8021 (See table below)
   - Other than basic DSP driver support, functional driver support for listed incremental functions are not supported.

Supported Speech Coder	Typical Application
G.723.1 6.3 & 5.3 kbps	H.324 for video conferencing over dial-up V.34 modems H.323 multi-media conferencing over LANs and TCP/IP type "packet-data" networks (e.g., Internet)
G.728 16 kbps	H.320 for video conferencing over ISDN
TrueSpeech 4.8 & 4.8/4.1 kbps	Proprietary low bit rate coder
16-bit or 8-bit linear 128 or 64 kbps	Microsoft WAVE files
G.711 Mu-Law/A-Law 64 kbps	Standard speech compression used in digital telephony systems (e.g., T1/E1)
Downloadable Speech Coders	
TrueSpeech 8.5 8.5 kbps	DSP Group speech coder built into Microsoft Windows 95. Also used in some DSVD modems and Internet Telephones
G.729A+B 8 kbps	Optional speech coder used in H.323
G.722 64 Kbps	7 KHz Wide bandwidth ADPCM audio coder used in H.320 (ISDN)

3. **Line Two Features**
   - Dedicated V.90 Modem interface (2-line scenario)
   - Isolated PSTN interface
   - Type 1 - Caller ID

Telephony Block Diagram



We claim:

1. A method of providing a multiple smart card simulator comprising:

(i) storing a first set of characteristics of a first smart card;

(ii) creating a first smart card simulator simulating operation of said first smart card;

(iii) storing a second set of characteristics of a second smart card;

(iv) simulating operation of said second smart card thereby creating a second smart card simulator;

(v) receiving a first input from a first smart card application;

(vi) modifying said first input based on said first set of characteristics;

(vii) sending a first modified input to said first smart card simulator;

(viii) modifying said first input based on such second set of characteristics to form a second modified input; and sending the second modified input to said second smart card simulator;

(ix) receiving a first output from said first smart card simulator generated in response to said first modified input;

(x) modifying said first output based on said first set of characteristics to form a first modified output;

(xi) transmitting said first modified output to said smart card application;

(xii) receiving a second output from said second smart card simulator in response to said second modified input;

(xiii) modifying said second output based on said second set of characteristics to form a second modified output; and

(xiv) transmitting said second modified output to said smart card application.

2. The method of claim 1 where said characteristics include one of the smart cards memory size, command sequence, error handling routine or quirks.

3. The method of claim 1 further comprising the steps:

(i) receiving a second input from a second smart card application;

(ii) modifying said second input based on said first set of characteristics;

(iii) sending a third modified input to said first smart card simulator;

(iv) receiving a third output from said first smart card simulator in response to said third modified input;

(v) modifying first said third output based on said first set of characteristics;

(vi) transmitting said third modified output to said second smart card application.

4. A method of simulating a multiple smart card environment comprising:

(i) detecting a simulated smart card insertion (step 400);

(ii) determining characteristics of the inserted card (step 405);

(iii) notifying registered listeners (step 410);

(iv) determining available applications (step 420);

(v) downloading to a simulated card and to a simulated terminal at least one smart card application (step 430) and

(vi) repeating (steps 400 - 430).

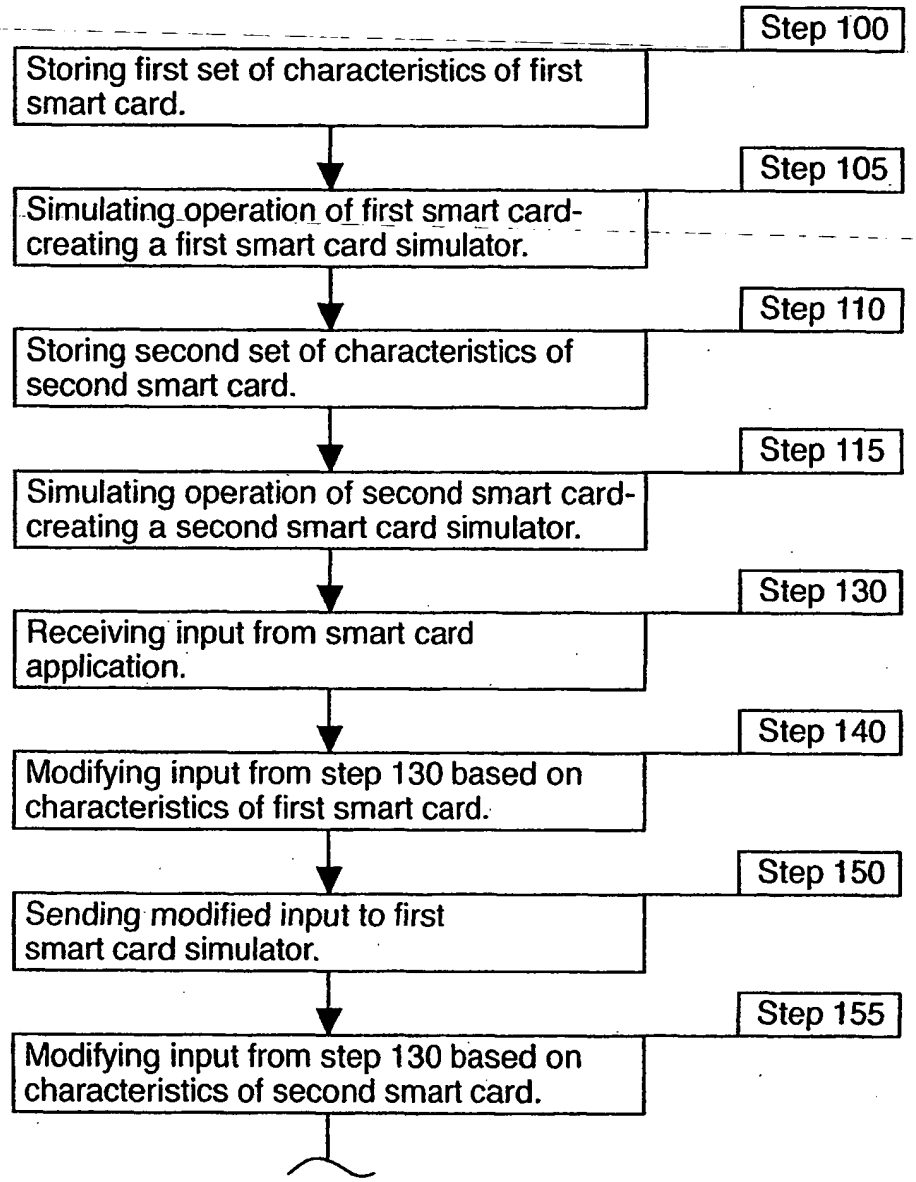

Step 100
Storing first set of characteristics of first smart card.
Step 105
Simulating operation of first smart card-creating a first smart card simulator.
Step 110
Storing second set of characteristics of second smart card.
Step 115
Simulating operation of second smart card-creating a second smart card simulator.
Step 130
Receiving input from smart card application.
Step 140
Modifying input from step 130 based on characteristics of first smart card.
Step 150
Sending modified input to first smart card simulator.
Step 155
Modifying input from step 130 based on characteristics of second smart card.

Figure 1A

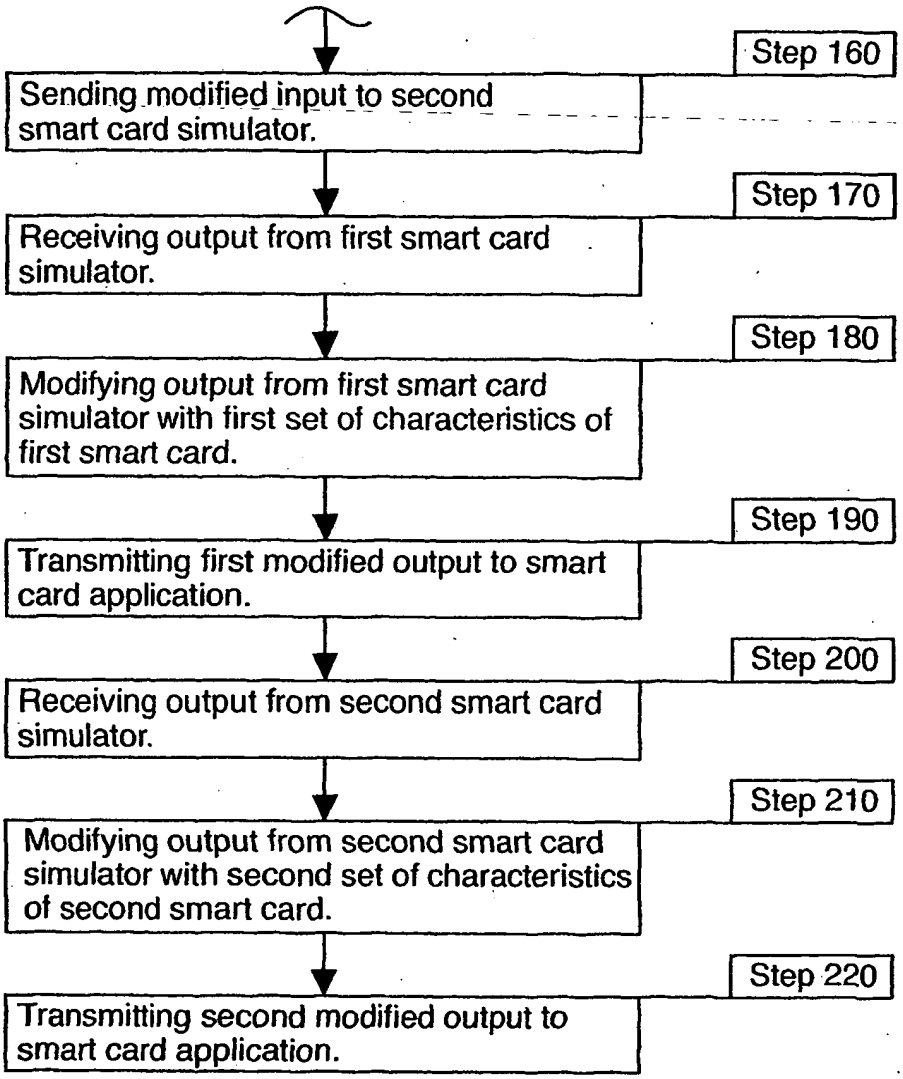

Step 160
Sending modified input to second smart card simulator.
Step 170
Receiving output from first smart card simulator.
Step 180
Modifying output from first smart card simulator with first set of characteristics of first smart card.
Step 190
Transmitting first modified output to smart card application.
Step 200
Receiving output from second smart card simulator.
Step 210
Modifying output from second smart card simulator with second set of characteristics of second smart card.
Step 220
Transmitting second modified output to smart card application.

Figure 1B

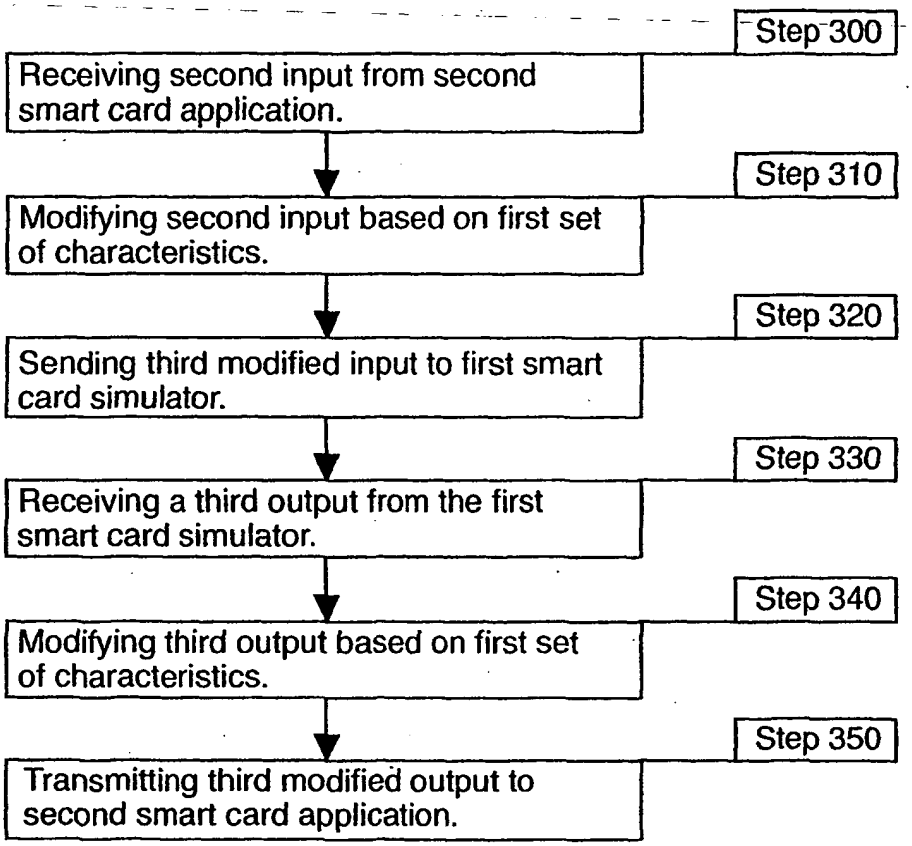

Step 300
Receiving second input from second smart card application.
Step 310
Modifying second input based on first set of characteristics.
Step 320
Sending third modified input to first smart card simulator.
Step 330
Receiving a third output from the first smart card simulator.
Step 340
Modifying third output based on first set of characteristics.
Step 350
Transmitting third modified output to second smart card application.

Figure 2

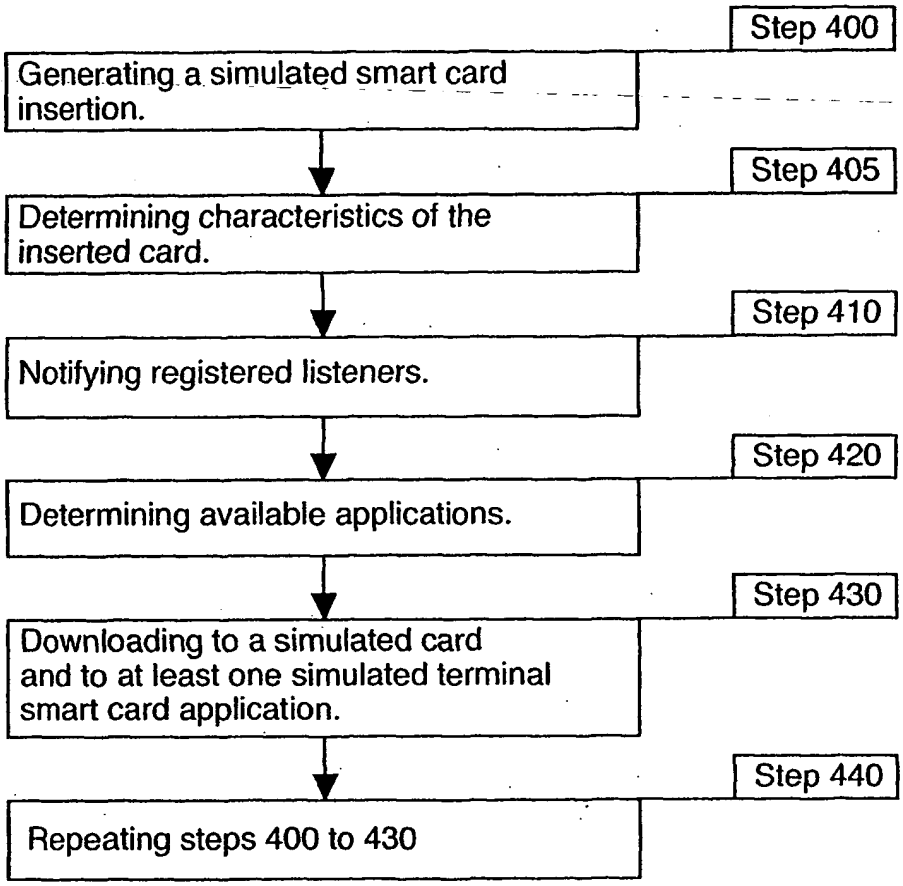

Step 400
Generating a simulated smart card insertion.
Step 405
Determining characteristics of the inserted card.
Step 410
Notifying registered listeners.
Step 420
Determining available applications.
Step 430
Downloading to a simulated card and to at least one simulated terminal smart card application.
Step 440
Repeating steps 400 to 430

Figure 3

INTERNATIONAL SEARCH REPORT

Intern[illegible]al Application No
PCT/CA 00/00703

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G07F7/10 G06F11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G07F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 98 25239 A (STRATEGIC ANALYSIS INC) 11 June 1998 (1998-06-11) page 2, line 35 -page 3, line 17 page 4, line 12 - line 26 page 5, line 7 - line 29 page 8, line 27 -page 9, line 8 page 10, line 12 -page 11, line 9	1-4
X	FR 2 667 419 A (GEMPLUS CARD INT) 3 April 1992 (1992-04-03) page 2, line 15 -page 4, line 1	1-4
A	PATENT ABSTRACTS OF JAPAN vol. 014, no. 141 (P-1023), 16 March 1990 (1990-03-16) & JP 02 005191 A (FUJITSU LTD), 10 January 1990 (1990-01-10) abstract	

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
6 November 2000	15/11/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Wolles, B

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Intern[redacted]al Application No
PCT/CA 00/00703

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9825239 A	11-06-1998	AU 5595398 A EP 0943136 A	29-06-1998 22-09-1999
FR 2667419 A	03-04-1992	NONE	
JP 02005191 A	10-01-1990	JP 2852381 B	03-02-1999